

SPECIAL PROLOG ISSUE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

101 March 1983

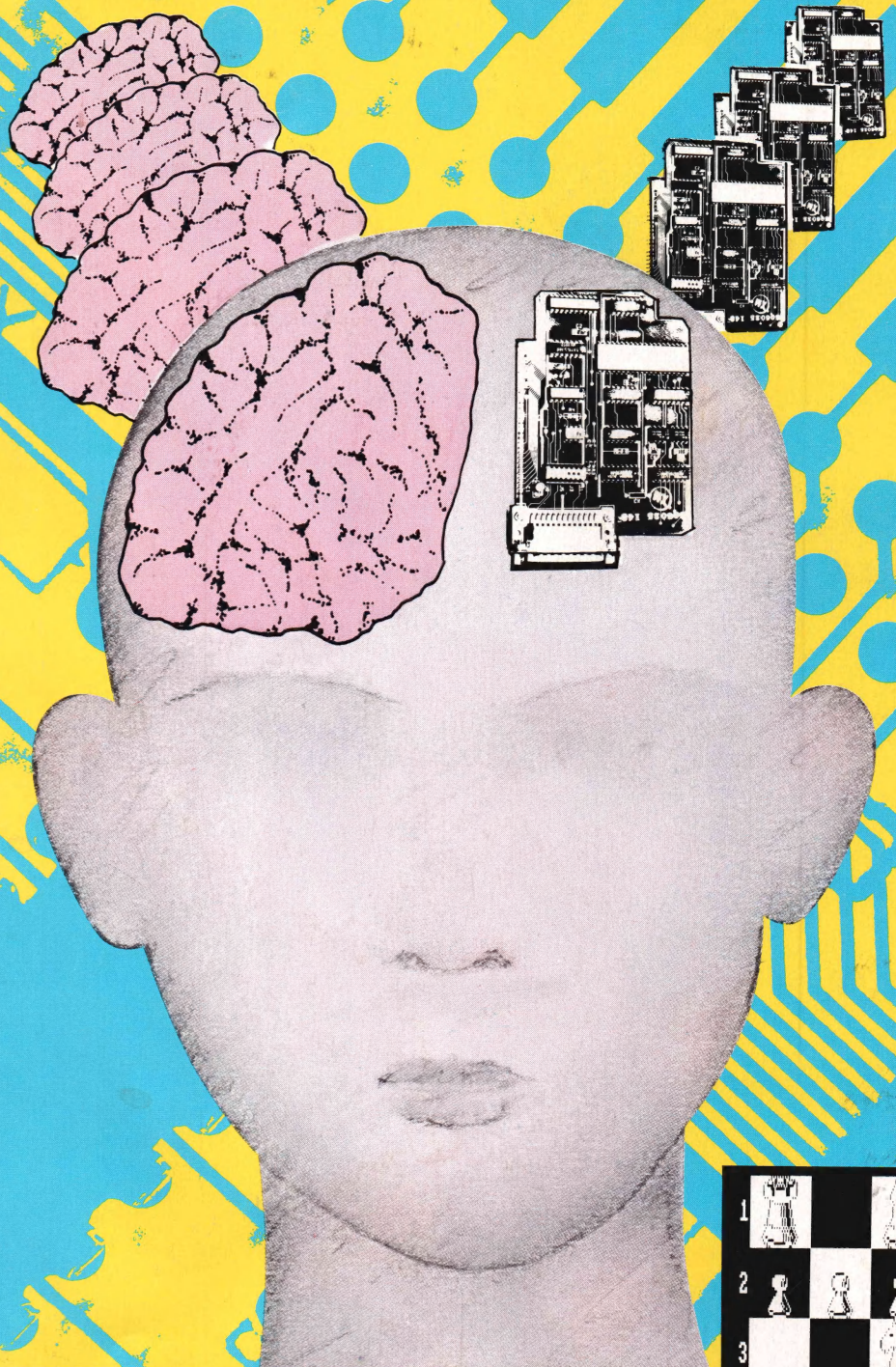
\$2.95 (3.50 Canada)

**How to Program
in Prolog**

**AI Competition
Winner**

**New Column:
The C Chest**

**A Proposed
Public-Domain
"Unix"**



SQUEEZE MORE OUT OF EVERY ON-LINE MINUTE.

WITH NEW VIDTEX™ COMMUNICATIONS SOFTWARE FROM COMPU SERVE.

Presenting the software package that makes your computer more productive and cost-efficient.

CompuServe's new Vidtex™ is compatible with many personal computers sold today (including Apple®, Commodore®, and Tandy/Radio Shack® brands). And it offers the following features*—and more—to let you communicate more economically with most time-sharing services (including CompuServe's Information Service).

Auto-Logon. Lets you log on to a host simply and quickly by utilizing prompts and responses defined by you. Also allows quick transmission of predefined responses to host application programs after logging on.

*Some versions of the Vidtex software do not implement all features listed.

Vidtex is a trademark of CompuServe, Incorporated. Apple is a trademark of Apple Computer, Inc. Commodore is a trademark of Commodore Business Machines. Radio Shack is a trademark of Tandy Corp.

Function Keys. Let you consolidate long commands into single keystrokes. Definitions can be saved to and loaded from disk file, allowing multiple definitions for multiple applications.

Error-Free Uploading and Downloading. CompuServe "B" Protocol contained in Vidtex lets you transfer from your computer to CompuServe and from CompuServe to your computer anywhere in the country. Also provides error-free downloading from CompuServe's extensive software libraries.

Full Printer Support. Printer buffer automatically buffers characters until printer can process; automatically stops on-line transmission when full; and automatically resumes transmission when capacity is re-established. Also, lets you print contents of textual video screen or RAM buffer at any time.

Capture Buffer. Saves selected parts of a session. Contents can be written to a disk file; displayed both on and off line; loaded from disk; and transmitted to the host.

On-line Graphics. Integral graphics protocol displays stock charts, weather maps and more.

If you are already a CompuServe subscriber, you can order Vidtex on line by using the GO ORDER command. Otherwise, check with your nearest computer dealer; or to order direct, call or write:

CompuServe

P.O. Box 20212, 5000 Arlington Centre Blvd.
Columbus, Ohio 43220

1-800-848-8199
In Ohio, call 614-457-0802

An H&R Block Company

Circle no. 23 on reader service card.



MPROLOG

The Programmer-Friendly Artificial Intelligence Language



In what is unquestionably one of the most dramatic breakthroughs in the entire field of Artificial Intelligence (AI), a new modular version of the logic based programming language Prolog is now available exclusively through Logicware.

A refined, programmer-friendly version of Prolog — the computer language selected as the basis for the Japanese Fifth Generation Computer System Project — MPROLOG has a wide and powerful range of business, industrial and research applications.

With MPROLOG you can:

- protect the corporate investment in expertise
- create efficient expert systems,
- shorten programming development times
- provide transportability of applications

Why MPROLOG Was Developed

It has long been a dream of system developers to create a programming language which could be used to describe a problem without having to program explicit steps towards a solution.

A program whose execution corresponds to a **controlled deduction** through facts, relationships and rules.

A program that will enable computers to learn, associate, draw conclusions and make decisions.

MPROLOG is that program language, the language of artificial intelligence.

When MPROLOG Can Be Used

MPROLOG can be used for:

- Expert Systems
- Deductive Databases
- Natural language understanding
- Computer-aided learning
- Fault diagnosis and repair
- Visual perception and guidance
- Intelligent Assistants
- Many areas of Artificial Intelligence

“Day-to-day use of MPROLOG... by programmers working with specialists... over the past eight years has produced some 30 applications spanning everything from architecture to pharmaceutical research.”

The Financial Times, London (U.K.)
November 13, 1983

MPROLOG Offers Seven Key Features

1. High performance with efficient use of resources
2. Hardware and operating system independence
3. Modular design capability; allows subsets of the problem to be specified and tested, offering substantial increases in productivity
4. Program development environment — Interactive Program Editor

- On-line “help”
- Concurrent editing and error correction
- Program Trace
- User-defined error handling
- Efficient garbage collection
- Over 250 built-in predicates

5. Interfaces to procedural languages and database managers

6. On-going support and enhancements

7. Comprehensive documentation and education

MPROLOG Works In A Flexible Operating Environment

The MPROLOG system is available on IBM mainframe computers under VM/CMS and MVS/TSO and Digital Equipment's VAX/VMS and VAX UNIX. Versions for the “super-micros” based on the M68000 CPU and the IBM PC-XT and AT are currently available with additional versions scheduled for release later this year.

To find out more about MPROLOG and Logicware call **1-714-476-3634**
(In Canada 1-416-665-0022)

or write

Logicware Inc., 5000 Birch St., West Tower
Suite 3000, Newport Beach
California, 92660

LOGICWARE

Boston • Toronto • Los Angeles

Circle no. 54 on reader service card.

Up Your AT™ for

\$56 per Megabyte!

■ While our specifications certainly speak for themselves, we thought you still might like to hear from some of our users:

■ "Emerald Systems expands the potential of PCs by providing the ability to access large amounts of data on line, quickly and reliably."
Terry Baptiste, Computerland, Lafayette, Ca.

■ "Service and support is great, which is an unusual experience. Emerald's software for backup and restore is invaluable. Can't put a price on it. Productivity and efficiency has increased at least 50%!"
Bruce Kittinger, Pinon Systems, Ft. Collins, Co.

■ Runs like a champ with 3-Com Ethernet."
Alvaro Ramirez, Micro Age, Miami, Fl.

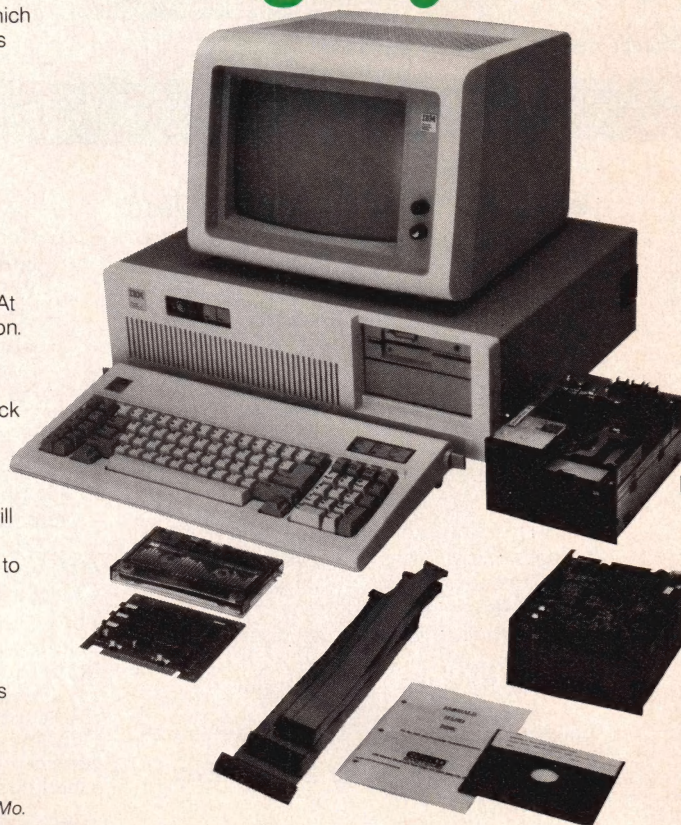
■ "... high capacity and flexibility. At last, a tape back up we can count on. And the price is right!"
John Acres, EDT, Las Vegas, Nv.

■ "The speed at which you can back up is very impressive."
Jim McEwen, Mercy Hospital, Portland, Me.

■ "When Emerald says your unit will be there on Thursday, it's there on Thursday! Delighted we were able to exceed the usual 32 megabyte restriction."
Steven Mayer, Take One Company, New York, N.Y.

■ "The Emerald 70 MB hard disk is extremely easy to install and work with. Emerald has a complicated piece of equipment made easy to use."
Tom Edler, Jewish Hospital, St. Louis, Mo.

■ "Our Emerald fixed disk installed quickly and easily. Emerald's reliable disk and tape backup further enhances LIBRA's high function accounting software."
Kenn White, Libra Programming, Salt Lake City, Ut.



HARD DISK

We've broken through the 32 MByte DOS barrier!

Now you can create up to 240 MByte databases on one file, with multiple volumes per disk drive. You get 14 times more storage with a 30% increase in access speed!

And that's not all:

Expands up to 280 MB for as little as \$56 per megabyte.
6 expansion slots
Up to 24 volumes
Drive sizes: 40, 70, 140
Internal or External
Also for PC, XT, and compatibles
Supports all PC compatible networks
Simple menu-driven installation

TAPE BACKUP

1/4" cartridge
1/2" 9 track
60 Megabytes
No lost data from tape run out
Backup and Restore Utility (BRU™)
software included
LAN Compatible *

***FREE APPLICATION GUIDE:**
"IBM LAN Installation and Implementation."

**Call (619) 270-1994,
or write to**

**Emerald™ Systems
Corporation**

**Mainframe Storage for Micros
4901 Morena Blvd,
San Diego, 92117
TLX 323458 EMERSYS
EASYLINK 62853804**

Distributed by Manchester Equipment of
NYC, and selected Entre, MicroAge and
Computerland stores.

Emerald, BRU, Up Your AT, and Mainframe Storage
for Micros are registered trademarks of Emerald
Systems Corporation. IBM PC/XT/AT are registered
trademarks of IBM Corporation.

EMERALD™

SYSTEMS CORPORATION

Mainframe Storage for Micros™

SOFTWARE

Computers Don't Live By Hardware Alone.

Without software, computers are little more than expensive bookends, but if I had to choose between using my computer as a bookend and running the latest version of *life* (in BASIC), my system would now be supporting a collection of Dr. Who catalogs. (After all there is more to computing than *life*.)

I started Micro Cornucopia as a forum for the adventurers who are designing and expanding new software tools. And over the years, Micro C has presented both serious and light-hearted looks at the trials everyone faces when probing the boundaries of a language or an algorithm.

In Micro C we have regular columns on Pascal (primarily Turbo), C, and FORTH. We also take close looks at operating systems, system monitors and new languages such as Modula II.

Public Domain Software

We put together public domain disks from the software submitted by Micro C readers. These disks contain formatters, screen dumpers, assemblers, disassemblers, text editors, and other useful goodies - many of which are available nowhere else. (Numerous CP/M80 and CP/M86 SIG/M programs originated within the Micro C group.)

An International Group

If you're interested in hardware and software — if you are ready to dig into your system and see what makes it tick — if you want to participate in an active international group that doesn't believe in black boxes — then this is your spot. Try us for a year: you'll see why Micro C has grown to over 80 pages of the most useful hardware and software information anywhere.

If you're looking beyond *life*, this is your spot.

David J. Thompson
Editor & Publisher

Risk Free Special Offer!

You get your 7th issue free when you order a year subscription (6 issues) for \$16. Plus, if you're not delighted with Micro C after receiving your first issue, just drop us a note and we'll refund your entire \$16, no questions asked.

Computers Don't Live By Software Alone.

Many computer publications seem to have forgotten that software has to run on something. That something is called hardware (see, we're ahead of them already). Those magazines that do remember, subscribe to the black-box hardware theory. "You don't open a black box, you just review it until the manufacturer goes bankrupt."

At Micro Cornucopia, we don't believe in black boxes. We've been building and expanding systems inside the pages of Micro C for nearly 4 years (a bit cramped, perhaps, but it's fun).

Microsystems and Micro C

Until recently, we've concentrated on single board systems — the Big Board, the Xerox, the Kaypro, and other CP/M Z80 systems (along with the 80186 based Slicer). However, with the demise of Microsystems, we've taken on support of homeless S-100 folk.

Dave Hardy has joined Micro C and his detailed coverage of the S-100 bus makes Micro C a primary source of information for this versatile system.

The Latest Processors

Here is where you'll hear about the upcoming micro-processors, as well as the current single board and S-100 board designs. Get the inside scoop on these systems and share the joys and frustrations we all face when parenting powerful new machines.

Coming Up

Through Micro C, you'll be able to design and build your own terminal, or turn that extra Xerox 820 board that's languishing in your drawer into a smart 64K printer buffer (just add a power supply and a ROM).

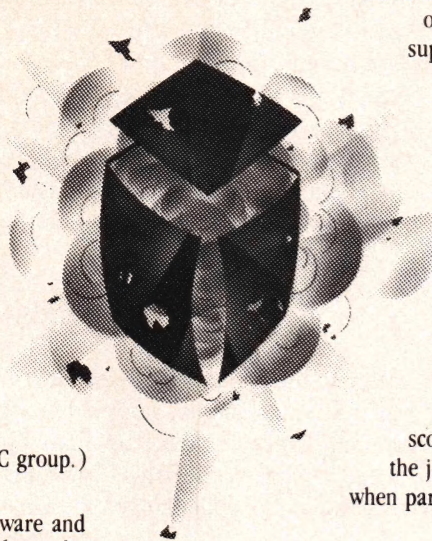
In fact, you'll get the greatest hardware projects for S-100 and single board systems as well as in-depth coverage of the great languages. About the only thing you won't get is a black box review. (But to get all this, you'll have to subscribe; you won't find Micro C in your local supermarket.)

Micro Cornucopia

P.O. Box 223 • Bend, OR 97709

Order # (503) 382-5060

HARDWARE



Exploding the Black Box Theorem.



Quantity	Description	Price	Total
	Year Subscription (plus 7th issue free)	\$16.00	
	Canada & Mexico	\$22.00	
	Other Foreign	\$30.00	
(US Funds only, payable on a US Bank)			

NAME _____

ADDRESS _____

CITY _____

STATE _____

ZIP _____

Only **\$16.00**

Make check or money order payable to:

Micro Cornucopia
Magazine

P.O. Box 223

Bend, OR 97709

☐ VISA

☐ MasterCard

Card No. _____

Exp. _____ Signature _____

CONTENTS

Editorial

Editor-in-Chief	Michael Swaine
Editor	Randy Sutherland
Managing Editor	Frank DeRose
Technical Editor	Alex Ragen
Editorial Assistant	Sara Noah
Contributing Editors	Robert Blum, Dave Cortesi, Ray Duncan, Allen Holub
Copy Editor	Polly Koch
Typesetter	Jean Aring

Production

Design/Production	
Director	Detta Penna
Art Director	Shelley Rae Doeden
Production Assistant	Alida Hinton
Cover	Claudia Majewski

Advertising

Advertising Director	Stephen Friedman
Advertising Sales	Walter Andrzejewski, Shawn Horst, Beth Dudas
DDJ Classifieds	Alex Williams
Advertising Coordinators	Alison Milne, Lisa Boudreau

Circulation

Fulfillment Manager	Stephanie Barber
Direct Response	
Coordinator	Maureen Snee
Promotions Coordinator	Jane Sharninghouse
Circulation Assistant	Kathleen Boyd

Administration

Finance Manager	Sandra Dunie
Business Manager	Betty Trickett
Accounting Coordinator	Mayda Lopez-Quintana
Accounts Payable	Denise Giannini
Billing Coordinator	Laura Di Lazzaro

M&T Publishing, Inc.

Chairman of the Board	Otmar Weber
Director	C.F. von Quadt
President	Laird Foshay

Entire contents copyright © 1985 by M&T Publishing, Inc. unless otherwise noted on specific articles. All rights reserved.

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, 2464 Embarcadero Way, Palo Alto, CA 94303. **ISSN 0278-6508**

Subscription Rates: \$25 per year within the United States, \$44 for first class to Canada and Mexico, \$62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank.

Foreign Distributors: ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progreso (France).

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

Prolog

A year ago when we were still being published by the old *People's Computer Company* in Menlo Park, I asked Mike Swaine if Artificial Intelligence was, by definition, beyond the scope of microcomputers. What resulted from that Sunday afternoon brainstorming session was the AI programming competition and this special issue.

As the programs and manuscripts rolled in we found ourselves building the issue around Prolog and we wished for a public domain tiny-Prolog to publish. We haven't found that yet, but we did get some very good Prolog pieces. Future issues of *DDJ* will include responses from Lisp people.

Speaking of future issues: June is the special telecommunications issue, as well as the copy deadline for the annual algorithms issue which will run in September. The annual Forth issue which usually runs in September will appear in October. Please submit Forth material by the Fourth of July.

Reader Ballot Results

Despite the popularity of our December 1984 focus on Unix, the article voted best in the Unix issue was not a Unix piece. Henry Seymour's "An Introduction to Parsing" received twice as many "best article" votes as the second most popular article, Alan Walworth's "Varieties of Unix." By the way, may we assume that because nobody challenged the Unix family tree on page 35, we got it right?

Mac Erratum

In the January Fatten Your Mac article, there is a discrepancy between the text on page 20 and Figure 1 on page 23. The figure correctly shows pin 8 soldered, but the text refers to pin 7 of the memory select IC. Readers who have called to point out the error inform us that the error in the text is obvious to anyone who works through and understands the article and the figures. We hope so.

This Month's Referees

Dave Cortesi, Resident Intern
Neil Jacobstein, Teknowledge
Scott Mace, Free-lance Computer Writer
Paul Reinholdtsen, Stanford University
Pekka Sinervo, Stanford Linear Accelerator

Randy Sutherland
Editor

Dr. Dobb's Journal

ARTICLES

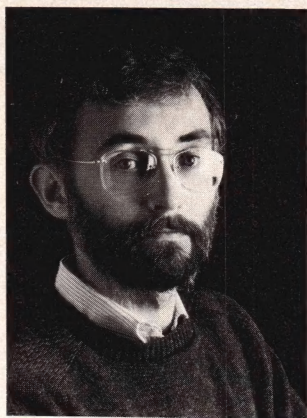
- | | | |
|---|--|--|
| <p>Programming in Logic
<i>by John Malpas</i></p> <p>Tour of Prolog
<i>by D. E. Cortesi</i></p> <p>Tax Advisor: A Prolog Program Analyzing Tax Issues
<i>by Dean Schlobohm</i></p> | <p>36</p> <p>44</p> <p>64</p> | <p>The development of logic programming from Frege to ICOT; features and applications of Prolog; implementations of Prolog. (Reader Ballot No. 192)</p> <p>A description of Prolog's style from a programmer's point of view. (Reader Ballot No. 193)</p> <p>The winner of the AI programming competition: this Prolog program assists attorneys in analyzing the tax rules in Section 318 (a) and explains its questions and conclusions. (Reader Ballot No. 194)</p> |
|---|--|--|

COLUMNS

- | | | |
|---|--|---|
| <p>C Chest
<i>by Allen Holub</i></p> <p>Realizable Fantasies
<i>by Richard Stallman</i></p> <p>16-Bit Software Toolbox
<i>by Ray Duncan</i></p> <p>CP/M Exchange
<i>by Bob Blum</i></p> | <p>10</p> <p>30</p> <p>96</p> <p>108</p> | <p>This new column takes over where the "C/Unix Programmer's Notebook" left off; this month, root module modifications for the Aztec CII compiler. (Reader Ballot No. 190)</p> <p>Richard Stallman outlines a proposal for GNU, a public domain Unix-like operating system. (Reader Ballot No. 191)</p> <p>A Z-8001 CPU for the S-100 bus; Mac notes; a fix for the 80286 BOUNDS instruction; DOS 3.0 differences and bugs; 8086 assembly language subroutines. (Reader Ballot No. 195)</p> <p>An RSX to correct the handling of random read errors under CP/M Plus; a PIP patch for CP/M Plus. (Reader Ballot No. 196)</p> |
|---|--|---|

DEPARTMENTS

- | | | |
|--|---|--|
| <p>Editorial</p> <p>Letters</p> <p>Reviews</p> <p>Of Interest</p> <p>DDJ Classifieds</p> <p>Advertiser Index</p> | <p>6</p> <p>8</p> <p>116</p> <p>122</p> <p>126</p> <p>128</p> | <p>Excalibur Technologies' SAVVY PC, Version 4.0.</p> <p>(Reader Ballot No. 197)</p> |
|--|---|--|



In one of my favorite novels the very existence of a certain island seems to depend on the direction from which you approach it: sort of a human-scale quantum uncertainty effect. Early in the novel a philosopher of science and a rather unimaginative psychologist discuss the island over lunch.

"The island doesn't bother you, does it, Dr. Blank," the philosopher of science says around a bite of toast, "because you look at it one side at a time."

One side at a time. Well, yes. That's a fundamental human limitation: we can only see one side of anything at a time. It comes of having our eyes set so close together. But that limitation shouldn't keep us from paddling around the island to discover how things look from the other side.

I recently spoke before several hundred engineers at a major electronics research center. Roughly half of them had within the past year been deeply involved in a project that, to their chagrin, the company ultimately decided would not become a product. The company had probably acted in its own best economic interest in putting the kibosh on the project, although some of the engineers seemed to think the decision might have been a mistake. If the decision was not in the company's own self-interest, then of course it was a mistake. Those are the alternatives.

Well, maybe in that case. But sometimes companies—and people—deliberately act contrary to their own perceived economic self-interest. It's generally assumed that, other things being equal, some money is better than no money, and we expect others to choose some over none whenever they perceive that they have such a choice. Corporate strategies are built on such an assumption about competitors' motives. But the assumption is not always warranted in fact.

Say, for example, that a bright programmer, convinced that software should be free, sets out to develop, with a little help from his friends, a Unix-compatible operating system and subsequently distributes it free to anyone who wants it. That's not a hypothetical example; it's exactly what Richard Stallman is proposing in his "GNU Manifesto," this month's Realizable Fantasy.

Say you're one of the hordes of people who have an economic interest in the success of Unix. If Stallman's fantasy is realized, you will soon face a competitor who is giving away his product, a strategy that some might think gives him a competitive edge. What do you do?

The fantasized confrontation between Unix and GNU could occur in any part of the software market. The number of programmers who could write a word processor or spreadsheet program better than many currently on the market grows daily. Large, complex programs are not exempt from such competition: as Stallman points out, modularizing the task solves the mere size problem, and a little reverse engineering unravels much complexity.

The fact is, both commercial software (*pace* Richard Stallman) and free software—written and distributed for fun or for pride of accomplishment or for philosophical reasons—are here to stay. You can't keep people from writing programs. But free software isn't going to destroy the market for commercial software; programs are too complex to be evaluated strictly on the basis of price.

On the other hand, smart commercial software developers will keep themselves aware of what free software exists, and not borrow trouble by trying to compete with free products. If they are really smart, they will also paddle around to the other side and *imagine* what free software might be written.

What is *Dr. Dobb's* view on the issue? We're in favor of good software, free and commercial, and have been promoting both kinds since 1976. I think we see both sides.

But not at the same time.

Michael Swaine

Michael Swaine

DIGITAL RESEARCH COMPUTERS

(214) 225-2309



IBM PC ADD ON BOARDS NOW IN STOCK!

RIO PLUS II™ — Multi-function board with 2 RS232 serial ports (1 standard, 1 optional), parallel I/O port, game port and clock/calendar. Comes with 64K memory (expandable to 384K) and PC Accelerator.

STB-RIO + II 64 List \$395 **\$286.00**

GRAPHIX PLUS II™

Multi-function video board featuring monochrome text, full screen monochrome graphics, RGB color, composite b&w video, a parallel port, a light pen interface, and an upgradable clock option.

STB-GRAX List \$495 **\$363.00**

SUPER RIO™

RAM-I/O multi-function board with 64K memory (upgrad-able to 768K), two serial ports, parallel printer port, game port and clock/calendar.

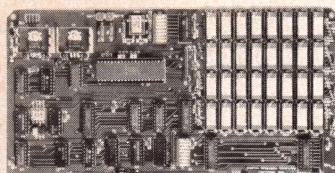
STB-SRIO-64 List \$419 **\$294.00**

All above boards come with "PC Accelerator" which is a RAM Disk Emulator and Printer Spooler that vastly increases your processing speed. All boards fully assembled and tested, and are warranted by the manufacturer for 1 year!

SPECIAL OFFER: Buy any two or more of the above boards and take an additional \$10 per board discount

256K S-100 SOLID STATE DISK SIMULATOR!
WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



BLANK PCB
(WITH CP/M™ 2.2
PATCHES AND INSTALL
PROGRAM ON DISKETTE)
\$69.95

(8203-1 INTEL \$29.95)

- FEATURES:**
- * 256K on board, using + 5V 64K DRAMS.
 - * Uses new Intel 8203-1 LSI Memory Controller.
 - * Requires only 4 Dip Switch Selectable I/O Ports.
 - * Runs on 8080 or Z80 S100 machines.
 - * Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
 - * Provisions for Battery back-up.
 - * Software to mate the LS-100 to your CP/M™ 2.2 DOS is supplied.
 - * The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
 - * Compare our price! You could pay up to 3 times as much for similar boards.

\$229.00

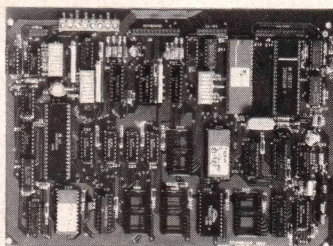
#LS-100 (FULL 256K KIT)

THE NEW ZRT-80 CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.

FEATURES:

- * Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
- * RS232 at 16 BAUD Rates from 75 to 19,200.
- * 24 x 80 standard format (60 Hz).
- * Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
- * Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
- * Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
- * 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
- * Composite or Split Video.
- * Any polarity of video or sync.
- * Inverse Video Capability.
- * Small Size: 6.5 x 9 inches.
- * Upper & lower case with descenders.
- * 7 x 9 Character Matrix.
- * Requires Par. ASCII keyboard.



BLANK PCB WITH 2716
CHAR. ROM, 2732 MON. ROM
\$49.95

SOURCE DISKETTE - ADD \$10
SET OF 2 CRYSTALS - ADD \$7.50

\$99.95

WITH 8 IN.
SOURCE DISK!
(CP/M COMPATIBLE)

(COMPLETE KIT,
2K VIDEO RAM)

64K S100 STATIC RAM

\$159.00
KIT

NEW!

LOW POWER!

150 NS ADD \$10

BLANK PC BOARD
WITH DOCUMENTATION
\$49.95

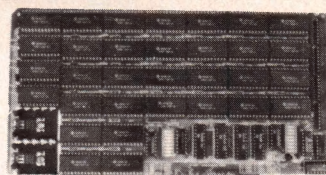
SUPPORT ICs + CAPS
\$17.50

FULL SOCKET SET
\$14.50

FULLY SUPPORTS THE
NEW IEEE 696 S100
STANDARD
(AS PROPOSED)

FOR 56K KIT \$145

ASSEMBLED AND
TESTED ADD \$50



FEATURES: PRICE CUT!

- * Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- * Fully supports IEEE 696 24 BIT Extended Addressing.
- * 64K draws only approximately 500 MA.
- * 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- * SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
- * 2716 EPROMs may be installed in any of top 48K.
- * Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
- * Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
- * BOARD may be partially populated as 56K.

64K SS-50 STATIC RAM

\$149.95
(48K KIT)

NEW!

LOW POWER!

RAM OR EPROM!

BLANK PC BOARD
WITH
DOCUMENTATION
\$52

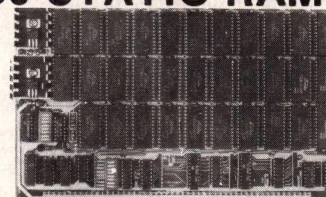
SUPPORT ICs + CAPS
\$18.00

FULL SOCKET SET
\$15.00

56K Kit \$169

64K Kit \$199

ASSEMBLED AND
TESTED ADD \$50



FEATURES:

- * Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
- * Fully supports Extended Addressing.
- * 64K draws only approximately 500 MA.
- * 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
- * Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
- * 2716 EPROMs may be installed anywhere on Board.
- * Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
- * One Board supports both RAM and EPROM.
- * RAM supports 2MHZ operation at no extra charge!
- * Board may be partially populated in 16K increments.

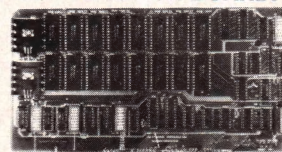
32K S100 EPROM/STATIC RAM

NEW!

FOUR FUNCTION BOARD!

NEW!

EPROM I
FULL
EPROM KIT
\$69.95
A&T EPROM
ADD \$35.00



BLANK
PC BOARD
WITH DATA
\$39.95

SUPPORT
ICs +
PLUS CAPS
\$16

FULL
SOCKET SET
\$15

We took our very popular 32K S100 EPROM Card and added additional logic to create a more versatile EPROM/RAM Board.

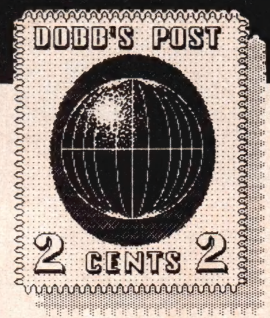
- FEATURES:**
- * This one board can be used in any one of four ways:
 - As a 32K 2716 EPROM Board
 - As a 32K 2732 EPROM Board (Using Every Other Socket)
 - As a mixed 32K 2716 EPROM/2K x 8 RAM Board
 - As a 32K Static RAM Board
 - * Uses New 2K x 8 (TMM2016 or HM6116) RAM's
 - * Fully Supports IEEE 696 Buss Standard (As Proposed)
 - * Supports 24 Bit Extended Addressing
 - * 200 NS (FAST!) RAM's are standard on the RAM Kit
 - * Supports both Cromemco and North Star Bank Select
 - * Supports Phantom
 - * On Board wait State Generator
 - * Every 2K Block may be disabled
 - * Addressed as two separate 16K Blocks on any 64K Boundary
 - * Perfect for MP/M™ Systems
 - * RAM Kit is very low power (300 MA typical)

32K STATIC RAM KIT — \$109.95
For RAM Kit A&T — Add \$40

TERMS: Add \$3.00 postage. We pay balance. Orders under \$15 add 75c handling. No. C.O.D. We accept Visa and MasterCard. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over \$50, add 85¢ for for insurance.

Digital Research Computers

P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309



Fantasies

Dear *DDJ*:

I read with interest your piece, "A Realizable Fantasy," in the November [1984] issue. Clearly, a product such as you describe would be potentially very useful as a productivity tool. A close approximation to such a tool may already exist!

Recently, on behalf of a client, we did a brief evaluation of a package called the Excelerator from Index Technology Corporation. The product—software + hardware—does exist and seems to function pretty much as described in the brochure. It's a very powerful utility and at least it's a start in the direction you outlined.

Not surprisingly, however, our client balked at the \$9,500 price tag—nearly twice what his XT cost! I agree that in a micro environment this is an impossible price, but there is no reason to suppose that such a capability can't be built and offered at substantially less. I hope someone does it soon.

Sincerely,
R. A. Langevin
7621 Fontaine St.
Potomac, MD 20854

Dear *DDJ*:

Some humble fantasies: extra CP/M BDOS calls for full-screen I/O, multi-character key input (function keys) and a utility that translates a submit file into a .COM (or .CMD) file.

Nelson Richardson
740 West End Avenue #21
New York, NY 10025

BU Bits

Dear *DDJ*:

Ian Ashdown's BU program published in the January [1985, #99] *DDJ* is a fine contribution to the growing list of public domain programs that can per-

form incremental backup given correct operation of the CP/M "archive bit" (t3'). As the author of one of the best known commercial programs for performing the same function, I feel I should comment on a serious factual error in Ian's article. He states: "... if you use a disk utility to set t3-prime to true in the file's directory entry, you will find that the BDOS resets it to false (zero) whenever the directory entry is changed. Since this means that the file has been opened, written to, and closed (or else renamed) by the BDOS, t3-prime is apparently an undocumented attribute bit that indicates when a file has been updated." I'm not sure where this idea got started, but for CP/M 2.2 it *just ain't so*. (Ian Ashdown is not the only notable to make this mistake: it also occurs on page 222 of David E. Cortesi's generally outstanding book *Inside CP/M*.) If you don't believe me, spend a few minutes with BU, set t3' on a file, change the file with a standard CP/M utility, and just see if the bit is reset.

Alas, while t3' is a functional archive bit under CP/M 3.0 and MPM, under CP/M 2.2 it acts just like any other attribute bit. There is, in fact, no code whatsoever in BDOS to reset this bit automatically when a file is changed, as those who have tried to implement Kelly Smith's original ARCHIVE.ASM program know full well. The BU program is almost useless under 2.2 unless BDOS is patched to install support for the archive bit. Our own product, Qbax, will function without patching for programs such as word processors and compilers that change files by rewriting them from scratch, but even our product requires a BDOS patch to catch random access updates in place. Such patches are extremely tricky, and while public domain software is wonderful it is also nice to have a company to rely on for

support when the patch doesn't work. Some of the versions of the BDOS patch for ARCHIVE.ASM that have appeared in various places have had some serious bugs.

Ian should have gotten a tip-off from his own comments: "Speaking of file copy utilities, a few are available that, under certain circumstances, will write a file without resetting its Archive attribute. One of these, oddly enough, is DRI's own PIP.COM." Not odd, just normal for CP/M 2.2! Vanilla PIP is not a rogue elephant that somehow bypasses BDOS (like so many programs that run under that other operating system, which shall remain nameless!) In fact PIP is written in a high-level language and uses orthodox BDOS calls. PIP fails to reset the archive bit because the facility to do so automatically in BDOS *doesn't exist*. The few programs available that do reset the archive bit do it as the result of explicit code placed there by the effort of their authors.

Best,
Jim Rosenberg
President
Amanuensis, Inc.
R.D. #1, Box 236
Grindstone, PA 15442
(412) 785-2806

Acknowledgments

Dear *DDJ*:

I want to set the record straight on three items that have arisen recently concerning authorship and propriety of certain copyrighted material.

Firstly, in the January 1985 issue of *DDJ*, page 67, Walt Piotrowski incorrectly gives me credit for the original MBOOT program published in *DDJ* October 1982 page 47. As the source listing of MBOOT in that article clearly indicates, Keith Petersen, W8SDZ, is the author of MBOOT and MBOOT is only

an adaptation of the original MODEM-ASM by Ward Christensen. This is an easy mistake for Walt to make and I doubt if anyone will be hearing from either Keith's or Ward's legal representatives.

Secondly, my dBASE II® command program DBSECURE.COM is a RUN-TIME® simulator, but requires the target environment to own *already* a licensed copy of dBASE II®. While DBSECURE will scramble and SCRUNCH command files like RUN-TIME®, it relies on DBASE.COM to execute the command file. This requirement is clearly indicated in the documentation.

Finally, NEWBASE.COM and NEWBASE.ASM will no longer support the SKIP-SIGNON-OPTION. The editors at DDJ brought it to my attention that such a feature could be used to by-pass the copyright notice of Ashton-Tate and that was not my intention.

As most authors for this magazine, I am available to try to answer questions and help solve problems with my published software. However, on behalf of all of us who put our name, address and phone number on the line, please call at reasonable hours (check time differences) and always include a self-addressed, stamped envelope when writing.

Gene Head
2860 NW Skyline Drive
Corvallis, Oregon 97330
(505)758-0279

DDJ

Are You In XTC™ Yet?

The Ultimate Programmer's Editor

Some folks have already discovered it. And they threw their other editors away! Why? Because XTC is incredibly powerful. It's also easy to learn, and easy to use. XTC has MORE editing facilities for LESS MONEY — 99 bucks



JUST LOOK AT THESE LUXURY FEATURES:

WINDOWS — 80 columns wide, independently 4-way scrollable, and non-overlapping. Define 'em the way you want to see them on your screen.

MACROS — Plenty of room for over 100 user-definable macro programs — you can assign 'em to function keys or labels up to 80 characters long!

KEYPAD EDITING — Standard where we come from. But for you mavericks, you can redefine those arrows to do auto-indenting, reformatting, the works! Need Wordstar compatibility? You can use your Wordstar editing commands here.

MULTITASKING — All of your macros can run in the foreground or independently in the background as separate processes while you continue editing.

CONTROL STRUCTURES — We've got everything, including IF THEN ELSE, WHILE, REPEAT, FOR, and BREAK.

EDITOR VARIABLES — Your macros can use plenty of variables to do just about anything. You get INTEGERS, BOOLEANS, and STRINGS, plus . . .

TEXT BUFFERS — More than you'll ever need — 20 in fact.

INTRODUCTORY OFFER

Want to compare XTC with your editor? Just ask for our demo disk (only \$5.00) and try it out. When you buy XTC, we'll knock five bucks off the price.

XTC outperforms any other programmable editor on all IBM /PC, /XT, and /AT computers (and true compatibles). XTC even works with your Sidekick and Turbo Pascal from Borland!

To get your copy of XTC now, order it over the phone — we can ship it the same day! Or, you can send in an order, just like this one:

XTC 99 bucks
Shipping and Insurance 3.50
Wash. res. add tax: 7.99
Want it COD? Add this 1.65
TOTAL IT UP, AND SEND IT QUICK!

These windows are really great! You can see several files at once — even different parts of the same file. That means you've got declarations in front of you while you're looking at the code that uses them!

Is XTC protected? Heck no! We even give you the source on a disk for your recreation — 7,000 lines of Pascal!

WENDIN™

Box 266 • Cheney, WA 99004 • USA • (509) 235-8088

Sidekick and Turbo Pascal are trademarks of Borland, International. Wordstar is a trademark of MicroPro. Wendin and XTC are trademarks of Wendin, Inc.

Pipes, Wild-Card Expansion, and Quoted Arguments

by Allen Holub

Hi. The C column is changing hands this month. So, before leaping into the thick of things, I want to talk a little about how I'd like the column to evolve. If you'd rather see this column do other things, write to me. I propose using this space for three major purposes:

(1) *Printing useful subroutines written in C.* This month's column is an example of the sort of subroutine I have in mind. In future columns, I'd like to talk about things like binary (AVL) tree management functions, operating system primitives (such as queue management routines), useful text manipulation primitives not found in the standard library (like the expression recognizer in *grep* discussed in *DDJ* No. 96), graphics routines, and so on. I've written compilers, operating systems, word processors, and the like, and I find myself using several pieces of these larger programs over and over again. It's those sorts of pieces that I'd like to print here. In addition to subroutines, complete small programs will appear here from time to time, such as an MSDOS version of the Unix utility "make," a simple sort utility, and the like.

Anything you're interested in is a potential topic, provided that you write and tell me what you want to know about. By the same token, if you have useful subroutines you'd like to share, please send them to me (on an IBM PC-compatible disk, if possible, or on paper, if not). While we're on the topic of participation, I've never written a bug-free program in my life. Consequently, although a program might run fine on my own system when compiled with my own compiler, I'm sure that unnoticed bugs creep in from time to time. If you find a bug, write and tell me; I'll print the correction as soon as possible. If you have a better way to do something and have a working program that does it (I'm not especially interested in theory), send it in.

(2) *Discussing technical and more theoretical issues.* Discussing some of the underlying theory of the C language and of compilers in general can help you understand a lot of the weirdness of the C language. By knowing about things such as stack frames and assembly language subroutine-calling conventions, you find that a lot of the hard-to-understand parts of the language become much clearer. This kind of knowledge also makes it easier to debug programs, so an occasional dip into theory could be useful.

(3) *Providing a general forum on C-related topics.* Want to start a public discussion on your own pet peeve? Send it in.

Adding Unix Command Line Functions

Unix provides an operating environment particularly helpful to programmers. Pipes, redirection, and wild-card expansion are all features that Unix itself provides, available to any program running under Unix regardless of the language in which that program was written. For example, when you say "grep something *.c" on the command line, Unix expands the *.c to a list of all files in the current directory that end in .c. By the time your main() function is executed, Unix will have filled argv with the names of these files rather than with the *.c. Similarly, when you say "foo > bar" or "foo | bar," Unix automatically takes care of sending the output of foo to the file bar (in the first case) or to the program bar (in the second).

Most operating systems aren't as nice. MSDOS does redirection, but, because it doesn't differentiate between stdout and stderr, most compilers can't let MSDOS actually do the redirection. MSDOS pipes are useable, but it doesn't do wild-card expansion. CP/M doesn't even do pipes. To make up for

these shortcomings, most manufacturers of C compilers implement some or all of these Unix functions inside the C program itself (rather than in the operating system where they belong). That's one reason why a two-line program can compile into 8K of code.

Unfortunately, most compiler manufacturers don't give you full Unix compatibility. I/O redirection almost always is included, and wild-card expansion almost always is omitted. Quoted arguments (which let you get imbedded spaces into a single argv entry) usually are not implemented either. This month, we will see how to add pipes, wild-card expansion, and quoted arguments to the Aztec CII compiler. It's not hard to modify these routines to work with many other compilers. In a future column, I'll give a version that works with Lattice C.

Root Modules

Before your main() subroutine is activated, a certain amount of housekeeping must be done. To accomplish this, most C compilers provide a subroutine that you usually don't see. This subroutine, called a root module, first does various initializations then calls your main() function. The root module for CP/M compilers often is inserted into your program automatically by the linker. MSDOS compilers typically require you to provide the name of the root module as the first filename argument submitted to link.

Among other things, the root module initializes the I/O system and parses the command line into the argv array. Redirection, wild-card expansion, and so forth are also done in the root module. So, to add these functions, all you have to do is modify the root. On the down side, anything you add to the root module will increase the size of every program you compile. It's often a good idea to have several

roots around—with varying complexity and size—and to use the one most suitable to a particular application.

Different manufacturers call root modules different things (Croot, root, _main, \$main, and so on). Sometimes there are two such functions. For example, the Computer Innovations compiler uses an assembly language routine \$main, which calls a C routine _main, which calls main(). Similarly, in Lattice C the assembly language routine c calls _main() (written in C), which in turn calls your main(). The root module for Aztec C called croot() is written in C.

Fortunately, the source for the root module usually is provided with the compiler, so you can modify it as necessary. The version of croot given here differs from the distribution version as follows:

(1) Any word on the command line containing the characters * or ? is assumed to be a filename containing wild-card characters. Croot will search the current user area on the indicated drive for any files matching the indicated name. The * and ? are expanded exactly the way they are in CP/M; that is, a * will match any character repeated zero or more times, and a ? will match any single character falling in the position in a name that corresponds to the position of the question mark. No character except a period may follow a *. The expanded list of files is passed through to main(). If you have three files on your disk with the extension .c, then the command line "program *.c" results in main() being passed an argc of 4; argv[1], argv[2], and argv[3] contain the names of the three files. The files are listed in no particular order.

(2) Any argument surrounded by double quotes ("arg") is treated as a single argument, even if there are embedded blanks in the string. For example, the command line:

```
program "this is one argument"
      "this is another"
```

enters main() with argc set to three. Argv[1] points at the string:

```
this is one argument
```

Argv[2] points at the string:

```
this is another
```

The quotes are stripped off for you. The wild-card characters * and ? are ignored if they are found in a quoted ar-

Still Fixing Bugs The Hard Way?



Ready to take the sting out of debugging? You can with Pfix86™ and Pfix86 Plus™, the most advanced dynamic and symbolic debuggers on the market today for PC DOS and MS-DOS™ programmers.

What other debugger offers you an adjustable multiple-window display so you can view program code and data, breakpoint settings, current machine register and stack contents all at the same time? And, an in-line assembler so you can make program corrections directly in assembly language. Plus, powerful breakpoint features that allow you to run a program at full speed until a loop has been performed 100 times, or have the program automatically jump to a temporary patch area.

Or maybe you're tired of searching through endless piles of listings for errors? With Pfix86 Plus you won't have to. You can

locate instruction and data by the symbolic name and using the symbolic address. Handle larger, overlaid programs with ease. And, Pfix86 Plus is designed to work with our Plink86™ linkage editor.

But that's not all. With a single keystroke you can trace an instruction and the action will be immediately reflected in code, data, stack, and register windows. Pressing a different key will elicit a special trace mode that executes call and loop instructions at full speed, as though only a single instruction were being executed.

And you get an easily accessible menu that makes the power of our debuggers instantly available to the new user, but won't inhibit the practiced user.

So, why struggle with bugs? Pfix86 by Phoenix. Pfix86 \$195. Pfix86 Plus \$395.

Call (800) 344-7200, or write.



Phoenix

Phoenix Computer Products Corporation

1416 Providence Highway, Suite 220
Norwood, MA 02062
In Massachusetts (617) 762-5030

Pfix86, Pfix86 Plus and Plink86 are trademarks of Phoenix Software Associates Ltd.
MS-DOS is a trademark of Microsoft Corporation

gument (i.e., they are passed through to `main()` as a * and a ?). If the quotes aren't present, `croot` will try to expand these characters into a filename.

(3) I/O redirection has not been modified from the Manx release version. The character `>` causes standard output to be "redirected" to the file whose name is immediately to the right of the character instead of to the console. The character `<` causes standard input to be taken from the file whose name is immediately to the left of the character instead of from the console. You have to be careful about using

pipes and redirection at the same time, though (see below).

(4) The pipe character (`|`) causes standard output from the program to the left of the character to be used as standard input to the program to the right of the character. Two temporary files called `$PIPE.IN` and `$PIPE.OUT` are created (on the A: disk in the current user area); they're automatically deleted when they are no longer needed. The program at the end of a pipe has to support redirection, but it need not support pipes. You may use more than one pipe on a single command line. For

example, the command line

```
grep -n foo | sort -d | lpr
```

executes `grep` first, invoking it with the arguments `-n` and `foo`. The output from `grep` is used as the input to the program `sort`, which is passed the argument `-d` as well. Finally, the output from `sort` is passed to the program `lpr`.

`Croot` does this chaining in two steps. It creates a temporary file in the first program in the chain, pretending that it saw a redirection directive (`>` `$pipe.out`) on the command line. Then when executing the function `exit()`, `croot` loads in the next program, using `execl()`, and executes it. `Croot` redirects the temporary file to the second program by adding a `<` `$pipe.in` to the command line sent to the second program (the name is changed before the command line is modified).

Given the above example, you could accomplish the same thing by typing:

```
grep -n foo > $pipe.out
ren $pipe.in = $pipe.out
sort -d < $pipe.in > $pipe.out
era $pipe.in
ren $pipe.in = $pipe.out
lpr < $pipe.in
era $pipe.in
```

If the temporary files can't be created, `croot` does not erase the `$pipe.in` file. However, it will attempt to abort any ongoing submit file by deleting the file `$$$sub`.

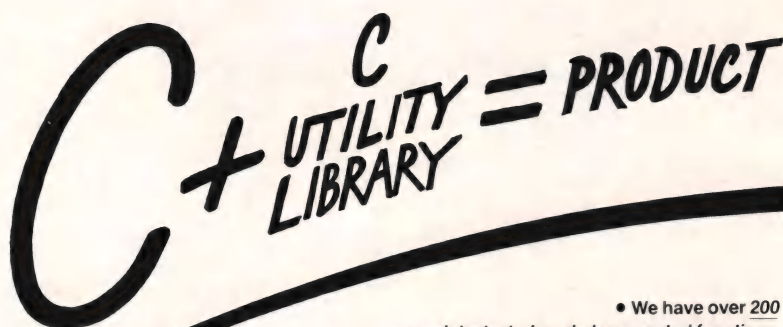
Pipes may be used along with redirection. For example, the command line `program1 <foo | program2 > bar` is perfectly legal. In this case, `program1` takes its input from the file `foo` and sends its output to `program2`. `Program2`, in turn, puts its own output into the file `bar`. You have to be careful, though: you can't direct output to two places at once or get input from two places at once. For example, "`program1 >foo | program2`" is illegal, as is "`program1 | program2 <foo`."

Support Routines

`Croot` requires several support routines; most of these are documented in Listing One (page 14). Some of these are useful enough to merit additional explanation.

do_wild

```
int do_wild(fname, argv, maxarg)
```



• We have over 200 complete, tested, and, documented functions. All source code and demo programs are included.

• The library was specifically designed for software development on the IBM PC, XT, AT and compatibles. There are no royalties.

• Over 95% of the source code is written in C. Experienced programmers can easily "customize" functions. Novices can learn from the thorough comments.

We already have the functions you are about to write
Concentrate on software development—not writing functions.

THE C UTILITY LIBRARY includes:

• Best Screen Handling Available • Windows • Full Set of Color Graphics Functions • Better String Handling Than Basic • DOS Directory and File Management • Execute Programs, DOS Commands and Batch Files • Complete Keyboard Control • Extensive Time/Date Processing • Polled ASYNC Communications • General DOS/BIOS gate • And More •

• The Library is compatible with: Lattice, Microsoft, Computer Innovations, Mark Williams and DeSmet.

C Compilers: Lattice C—\$349, Computer Innovations C86—\$329; Mark Williams C—\$449.

C UTILITY LIBRARY \$149

Order direct or through your dealer. Specify compiler when ordering. Add \$4.00 shipping for UPS ground, \$7.00 for UPS 2-day service. NJ residents add 6% sales tax. Master Card, Visa, check or P.O.



ESSENTIAL SOFTWARE, INC

P.O. Box 1003 Maplewood, New Jersey 07040 914/762-6605


```
char *fname, **argv;
int maxarg;
```

A general purpose directory-searching utility, `do_wild()` creates an `argv`-like array of pointers to strings, one string for each file in the current directory that matches the filespec "`fname`." It returns the number of valid elements in the `argv` array after the expansion; the return value can be used to update `argc`.

If `fname` contains no wild-card characters, only one name is expanded. If `fname` does not exist, nothing is done, and the routine returns a zero. If `fname` contains a `*` or a `?`, `argv` will contain a pointer to a string for each possible matching name. If the string begins with `x:` (where `x` may be replaced by any valid CP/M drive id), the indicated drive is searched instead of the current drive. On entry, the parameter `argv` should point at the beginning of an uninitialized array of character pointers. `Maxarg` is the maximum permitted size of `argv`.

execl

```
execl( name, arg1, ... argN,
(char *)0 )
char *name, *arg1 ... *argN;
```

`Execl()` chains to the program "`name`" by overwriting the current program and transferring control to the new program; the extension `.COM` is automatically appended onto "`name`."

`Execl()` has a variable number of arguments. `Arg1` through `argN` are concatenated and passed through to the new program as its command line. The rightmost argument in the list must be zero. Because of the way the next program is loaded, the length of all the concatenated `arg` strings may be at most 79 characters. Note that `execl()` inserts a space between each string. Any characters beyond the 79 are truncated.

`Execl()` should not return. If it does return, then `execl()` can't open the specified file. `Execl()` does not print an error message in this latter case. Note that the contents of the default FCB and the default DMA buffer (at addresses `0x5c` and `0x80`) are undefined if `execl()` returns. `Execl()` makes no attempt to close any files, so

be sure to close any opened files before the `execl()` call.

`Execl()` uses two assembly language files to load the next program in the pipe, "`loader`" and "`loadldr`" (Listing Two, page 27). `Loader` reads the next program into memory then transfers control. `Loadldr` puts the loader itself into memory.

initfcb

```
initfcb(filename, fcb)
char *filename, *fcb;
```

`Initfcb()` initializes the fcb (CP/M file control block) pointed at by "`fcb`" to the filename "`filename`." If the file-

name has `x:` as the first two characters of the string, the drive field is set to drive `x` as well; all other fields are filled with zeroes.

A `*` in the filename is expanded out to fill the rest of the field with `?`s. So, the wild-card characters `*` and `?` are treated as they are by CP/M. For example:

```
xxx.yyy becomes xxx yyy
xx*.y* becomes xx?????y??
" " or *. becomes ?????????
```

The filename argument may be terminated by either a `\0` or a space.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 190.

MEMO: C Programmers **QUIT WORKING SO HARD.**

These people have quit working so hard: IBM, Honeywell, Control Data, GE, Lotus, Hospitals, Universities & Government Aerospace.

THE GREENLEAF FUNCTIONS™

THE library of C FUNCTIONS that probably has just what you need ... **TODAY!**

... already has what you're working to re-invent
... already has over 200 functions for the IBM PC, XT, AT, and compatibles
... already complete ... already tested ... on the shelf
... already has demo programs and source code
... already compatible with all popular compilers
... already supports all memory models, DOS 1.1, 2.0, 2.1
... already optimized (parts in assembler) for speed and density
... already in use by thousands of customers worldwide
... already available from stock (your dealer probably has it)
... It's called the **GREENLEAF FUNCTIONS**.

Sorry you didn't know this sooner? Just order a copy and then take a break — we did the hard work. Already.

THE GREENLEAF FUNCTIONS GENERAL LIBRARY: Over 200 functions in C and assembler. Strength in DOS, video, string, printer, async, and system interface. All DOS 1 and 2 functions are in assembler for speed. All video capabilities of PC supported. All printer functions. 65 string functions. Extensive time and date. Directory searches. Polled mode async. (If you want interrupt driven, ask us about the **Greenleaf Comm Library**.) Function key support. Diagnostics. Rainbow Color Text series. Much, much more. **The Greenleaf Functions.** Simply the finest C library (and the most extensive). All ready for you. From Greenleaf Software.

... **Specify compiler** when ordering. Add \$7.00 each for UPS second-day air. MasterCard, VISA, check, or P.O.



- ◆ Compilers:
 - CI C86.....\$349
 - Lattice\$395
 - Mark Williams ...\$475
- ◆ General Libraries....\$175
 - (Lattice, Microsoft, Mark Williams, CI C86)
- ◆ DeSmet C.....\$150
- ◆ Comm Library.....\$160

GREENLEAF SOFTWARE, INC.

2101 HICKORY DRIVE ◆ CARROLLTON, TX 75006 ◆ (214) 446-8641

Circle no. 43 on reader service card.

C Chest (Text begins on page 10)

Listing One

```
/*                                     Croot.c
*
*                                     Wildcard expansion & pipes for Aztec CII
*
* This version of Croot.c has been extensively modified by A. I. Holub.
* Manx Software Systems is in no way responsible for this program.
*
* The portions of this program printed in boldface and are the original
* croot. They are Copyright (c) 1981, 1982 by Manx Software Systems and
* (c) 1982 by Thomas Fenwick.
*
* The portions of this program not printed in boldface are
* Copyright (c) 1985, Allen I. Holub. These portions may be reproduced
* for personal, non-profit, use only. All other use is prohibited.
*
* You should replace the croot in libc.lib with this module.
*/

#include "a:errno.h"
#include "a:fcntl.h"
#include "a:io.h"

#define MAXARGS          128                /* Maximum argc */
#define LOADERSIZE       49                /* Size of the loader in bytes */
#define DEF_FCB          0x5c             /* CP/M Default FCB location */
#define FCB_SIZE         36                /* Size of an FCB in bytes */

#define CBUF              ((char *)0x80)    /* CP/M command line buffer */
#define CBUFEND           (char *) (0x80+128-LOADERSIZE)

/*
* Various BDOS function calls:
*/

#define srchfirst(fcb)    ( bdos( 0x11, (fcb) ) & 0xff )
#define srchnext(fcb)     ( bdos( 0x12, (fcb) ) & 0xff )
#define bdosopen(fcb)     ( bdos( 0x0f, (fcb) ) & 0xff )

int badfd(), noper();

/* Channel table: relates fd's to devices
*/

struct channel chantab[] =
{
    { 2, 0, 1, 0, noper, 2 },
    { 0, 2, 1, 0, noper, 2 },
    { 0, 2, 1, 0, noper, 2 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
    { 0, 0, 0, 0, badfd, 0 },
};

static char *Argv[MAXARGS];
```

(Continued on page 16)

WIZARD C

Fast compiles, fast code and great diagnostics make Wizard C unbeatable on MSDOS. Discover the powers of Wizard C:

- ALL UNIX SYSTEM III LANGUAGE FEATURES.
- UP TO A MEGABYTE OF CODE OR DATA.
- SUPPORT FOR 8087 AND 80186.
- FULL LIBRARY SOURCE CODE, OVER 200 FUNCTIONS.
- CROSS-FILE CHECKS OF PARAMETER PASSING.
- USES MSDOS LINK OR PLINK-86.
- CAN CALL OR BE CALLED BY PASCAL ROUTINES.
- IN-LINE ASSEMBLY LANGUAGE.
- 240 PAGE MANUAL WITH INDEX.
- NO LICENSE FEE FOR COMPILED PROGRAMS.

The new standard for C Compilers on MSDOS!

Only \$450

WSS

For more information call (617) 641-2379
Wizard Systems Software, Inc.
11 Willow Ct., Arlington, MA 02174
Visa/Mastercard accepted

Circle no. 116 on reader service card.

Fortran Scientific Subroutine Package

Contains Approx. 100 Fortran Subroutines Covering:

- | | |
|----------------------------------|-----------------------------|
| 1. Matrix Storage and Operations | 7. Time Series |
| 2. Correlation and Regression | 8. Nonparametric Statistics |
| 3. Design Analysis | 9. Distribution Functions |
| 4. Discriminant Analysis | 10. Linear Analysis |
| 5. Factor Analysis | 11. Polynomial Solutions |
| 6. Eigen Analysis | 12. Data Screening |

Sources Included

\$295.00

FORLIB-PLUS™

Contains three assembly coded LIBRARIES plus support. FORTRAN coded subroutines and DEMO programs.

The three LIBRARIES contain support for GRAPHICS, COMMUNICATION, and FILE HANDLING/DISK SUPPORT. An additional feature within the graphics library is the capability of one fortran program calling another and passing data to it. Within the communication library, there are routines which will permit interrupt driven, buffered data to be received. With this capability, 9600 BAUD communication is possible. The file handling library contains all the required software to be DOS 3.0 PATHNAME compatible.

STRINGS & THINGS™

Support for CHARACTER MANIPULATION (string support), SHELL, BATCH, MUSIC, CMD LINE, and ENVIRON CTRL.

\$69.95 each

P.O. Box 2517
Cypress, CA 90630

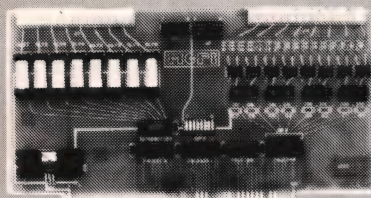


(714) 894-6808

California residents, please add 6% sales tax
Versions available for IBM Professional Fortran
or MICROSOFT 3.2 Fortran

Circle no. 1 on reader service card.

MULLEN S-100: Real Time, Real World CONTROLLER

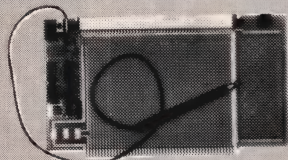


ICB-10 CONTROLLER BOARD
\$219, assembled and tested.

This 8 channel digital I/O controller can monitor inputs and control outputs. It features an easy to read manual that has schematics, component list, and programming examples as well as provocative insights on potential applications.

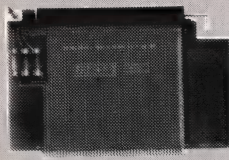
Examples of applications are included in a reprinted article that demonstrates two MULLEN CONTROLLER BOARDS in an interactive system that: feeds a cat; irrigates a garden dependent on soil moisture; closes the window when it rains; controls the thermostat for optimum comfort; controls appliances, lights, security system, and weather monitoring station (logging temperature, wind speed and direction, and graphing pollution content of the atmosphere.) Solenoids, microswitches, pneumatic actuators, pH sensors, and other devices are used in this system.

MULLEN S-100: DEBUGGERS



TB-4a EXTENDER BOARD
The latest in our TB line, the most widely used add-ons in the industry. Features logic probe, formed-lead edge connectors, pulse catcher switch and reset button.

\$89, assembled and tested.



ZB-1 ZIF EXTENDER BOARD
This debugger features Zero Insertion Force edge connectors for easy board changes and long life. Expect 2,000 or more insertions rather than the usual 300 to 400 with tension type connectors.

\$159, assembled and tested.



MULLEN COMPUTER PRODUCTS, INC.

AVAILABLE:

Priority One Electronics,
Chatworth, CA • (213) 709-5111
Jade Computer Products
Hawthorne, CA • (213) 973-7707
E² Computer Products
Hayward, CA • (415) 786-9203

Mullen Computer Products is the industrial distributor for CompuPro's products. For more information, call us at (415) 783-2868 or write MCPI, 2260 American Ave., #1, Hayward, CA 94545. OEM sales available from factory. Prices are subject to change without notice.

Circle no. 91 on reader service card.

C Chest

Listing One

(Listing Continued, text begins on page 10)

```
static char    Argbuf[128];
static int     Argc;

static int     Doingpipe = 0;
static char    *Cmdtail;

/*      File names for pipe processing. The padding lets us patch the name
 *      with DDT rather than re-compiling.
 */

char    *Pipe_in  = "A:$PIPE.IN\0\0\0\0" ;
char    *Pipe_out = "A:$PIPE.OUT\0\0\0\0" ;

/*-----*/

Croot()
{
    /*      Root module for Aztec CII. This is not the best C programming
    *      in the world. It uses gotos; what some of the code is doing
    *      is, by necessity, somewhat obscure. On the other hand it
    *      does a bunch of complex stuff relatively effciently.
    */

    register int k;
    register char *cp, *fname;

    blockmv(Argbuf, CBUF + 1, 127); /* Copy the command line somewhere */
    Argbuf[*CBUF & 0x7f] = 0;        /*      safe and null terminate it. */

    Argv[0] = "";                    /* Initialize argv and argc      */
    cp = Argbuf;
    Argc = 1;

    while (Argc < MAXARGS)
    {
        while (*cp == ' ')           /* Skip leading blanks      */
            ++cp;

        if (*cp == 0)                 /* At end of command line */
            break;

        else if( *cp == '|' )         /* Process a pipe */
        {
            Doingpipe = 1;
            Cmdtail = ++cp;

            /*
             *      Pretend we saw a ">" to the Pipe_out file.
             */

            fname = Pipe_out ;
            k = 1;
            goto redir2;

        }
        else if (*cp == '>')           /* redirect output */
        {
            k = 1;
            goto redirect;
        }
    }
}
```

(Continued on page 18)

Another in a series of
productivity notes on MS-DOS™
software from UniPress.

**Subject: Multi-window full
screen editor.**

Multiple windows allow several files
(or portions of the same file) to be
edited simultaneously. Program-
mable through macros and the built-
in compiled MLISP™ extension
language.

Features:

- Famed Gosling Version.
- Extensible through the built-in
MLISP programming language and
macros.
- Dozens of source code MLISP
functions; including C, Pascal and
MLISP syntax checking.
- EMACS runs on TI-PC™, IBM-PC AT™,
DEC Rainbow™ or any other MS-DOS
machine. Requires at least 384k.
- Run Lattice® C or PsMake™ in
the background and EMACS will
point to any errors for ease of de-
bugging. PsMake is a UNIX™-style
"make" utility to automate the pro-
cess of building complex programs.
- Optional Carousel Tools: UNIX-
like facilities including cat, cp, cd,
logout, ls, mv, pwd, rm, set, sh
and more.

Price:

EMACS	\$475
One month trial	75
Available for UNIX and VMS.	
PsMake	179
Carousel Tools	149
Full System	1,299
Includes EMACS (object), PsMake, Lattice C, PHACT™ ISAM and Carousel Tools.	

TEXT EDITING

UNIPRESS EMACS™

Subject: Compiler for MS-DOS.

Lattice C Compiler is regarded as
the finest compiler for MS-DOS and
is running on thousands of 8086
systems.

Features:

- Runs on the IBM-PC™ under
MS-DOS 1.0, 2.0 or 3.0
- Produces highly optimized code.
- Small, medium, compact and
large address models available.
- Standard C library.
- PLINK—optional full function
linkage editor including overlay
and support.

Price:

Lattice C Compiler	\$425
PLINK	425

COMPILER FOR THE 8086™ FAMILY

LATTICE® C COMPILER

**Subject: Powerful Keyed File
Access for MS-DOS.**

PHACT ISAM is a keyed B+ tree
file manager providing easy access
to and manipulation of records in
a database.

Features:

- Supports fixed and variable length
records (1-9999 bytes).
- Up to 9 alternate indices are sup-
ported.
- Record locking allows each record
in the database to allow multiple
simultaneous updates.
- Records can be accessed on full
or partial key.
- Includes full Lattice linkable library
and high-level functions.

Price:

PHACT ISAM	\$250
Source Code available, call for terms.	

For more information on these and
other UNIX software products, call or
write: UniPress Software, Inc., 2025
Lincoln Hwy., Edison, NJ 08817.
Telephone: (201) 985-8000. Order
Desk: (800) 222-0550 (Outside NJ).
Telex: 709418. Japanese Distributor:
Softec 0480 (85) 6565. European Dis-
tributor: Modulator SA (031) 59 22 22.

OEM terms available.
Mastercard/Visa accepted.

ISAM FILE SYSTEM

PHACT™

C Chest (Listing Continued, text begins on page 10)

Listing One

```

else if (*cp == '<')                /* redirect input */
{
    k = 0;
redirect:
    while (*++cp == ' ')
        ;

    fname = cp;
    while (*++cp)
    {
        if (*cp == ' ')
        {
            *cp++ = 0;
            break;
        }
    }

    close(k);
    k = k ? creat(fname, 0666) : open(fname, O_RDONLY) ;
    if (k == -1)
    {
        perr("Can't open file for redirection <");
        perr(fname);
        perr(">");
        exit(10);
    }

    if( Doingpipe )
    {
        /* The rest of the cmd tail is for the next
        * process so stop processing it.
        */

        break;
    }
}

else if (*cp == '"')
{
    Argv[Argc++] = ++cp;

    while ( *cp  && (*cp != '"') )
    {
        cp++;
    }
    *cp++ = 0;
}
else if ( haswild(cp) )
{
    Argc += do_wild(cp, Argv + Argc,  MAXARGS-Argc-1);
    goto skippast;
}
else
{
    Argv[Argc++] = cp;
skippast:
    while (*++cp)
    {
        if (*cp == ' ')

```

(Continued on page 20)



PRESENTS

CP/M[®]

FOR THE

Macintosh[™]

I.Q. SOFTWARE, in one historic move, brings more proven programs to the Macintosh than have existed to date. In the giant world of CP/M based software, **CP/M FOR THE MACINTOSH** delivers the same friendly stand-alone environment to the software developer (professional programmer). If you are wondering why so many developers know and use CP/M it is because CP/M is powerful, easy to learn and easy to use. So easy to use that almost 1 MILLION USERS have CP/M based systems using CP/M based programs and enjoy these same benefits that you will with **CP/M FOR THE MACINTOSH**.



CP/M is a registered trademark of Digital Research, Inc.
Macintosh is a registered trademark of Apple Computer, Inc.

2229 East Loop 820 North
Fort Worth, Texas 76118
(817) 589-2000


```

        {
            *cp++ = 0;
            break;
        }
    }
}
exit(main(Argc,Argv));
}

/*-----*/

execl( name, args )
char    *name, *args;
{
    /*      Execl has a variable number of arguments starting
    *      with args and continuing until a 0 arg is found.
    *      It is called with:
    *
    *          execl( filename, arg1, arg2 , ... , argN, 0 );
    *
    *      All arguments must be character pointers.
    *      Execl overlays the current program with the program
    *      named "name". All the args are concatenated together
    *      with the args separated from one another by spaces.
    *      The concatenated array is used as the command line
    *      for the new process. Execv adds the .com to the end of
    *      name.
    */

    register char    *cp, *dest, **argp;

    strcpy (CBUF, name );
    strcat (CBUF, ".COM" );
    initfcb(CBUF, DEF_FCB );           /* Try to open the file      */

    if( bdosopen(DEF_FCB) == 0xff )    /* Can't do it              */
        return(0);

    /*
    *      Put together a new command line from the arguments
    */

    dest = CBUF + 1;

    for(argp = &args ; *argp && dest < CBUFEND ; argp++ )
    {
        for(cp = *argp ; *cp && dest < CBUFEND ; *dest++ = *cp++ )
            ;

        *dest++ = ' ';
    }
    *dest = '\0';

    *CBUF = (int)dest - (int)CBUF ;

    loadldr();                        /* We won't return from loadldr */
}

```



```

}

/*-----*/

exit(code)
{
    /*      Finish up the execution of a C program. Chain to the
    *      next process if appropriate.
    */

    register char    *tailp, *src, *dest ;
    closall();                /* Close all open files      */
    unlink(Pipe_in);          /* Get rid of the pipe temp file */

    if ( code )                /* Try to abort an ongoing submit */
        unlink("A:$$$$.SUB"); /*      file.                  */

    if( Doingpipe )           /* Chain to next program in pipe */
    {
        rename(Pipe_out, Pipe_in);

        /* Get the file name from the command tail. First strip out
        * leading blanks, then terminate the file name with a null
        * if necessary (the rest of the command tail will be
        * arguments to the next file.
        */

        while(*Cmdtail == ' ')
            Cmdtail++;

        tailp = Cmdtail;

        while( *tailp && *tailp != ' ' )
            tailp++;

        if( *tailp )
            *tailp++ = '\0';

        /* Chain to the next program      */

        execl( Cmdtail, "<", Pipe_in, tailp, (char *)0 );

        /* If execl returns then we couldn't open the file */

        perr("Couldn't open file at end of pipe: ");
        perr(Cmdtail);
        perr(".COM");
        unlink("A:$$$$.SUB"); /* Try to kill a submit file */
        unlink("$$$$.SUB");
    }

    boot_();
}

/*-----*/

perr(str)
char    *str;
{
    /*      Print a string by brute force. We can use this
    *      whether or not stdout is open.
    */

    while(*str)
        bdos(2, *str++);
}

```


C Chest

(Listing Continued, text begins on page 10)

Listing One

```
}

/*-----*/

badfd()          /* Called when a bad file descriptor is found */
{
    errno = EBADF;
    return -1;
}

/*-----*/

noper()          /* Do nothing, return 0 */
{
    return 0;
}

/*-----*/

initfcb(filename, fcb)
char          *filename, *fcb;
{
    /*      Initialize the fcb pointed at by "fcb" with the filename
    *      pointed to by "filename". Filename may include a drive
    *      designator (X:) or either of the wild card characters * or
    *      ?. See the text for more details.
    */

    register char    *fp;
    register int     i;

    for ( i = FCBSIZE , fp = fcb ; --i >= 0 ; *fp++ = 0 )
        ;

    if( *filename && filename[1] == ':' )
    {
        /*      Fill in the drive field of the fcb
        */

        *fcb = (char)( toupper(*filename) - 'A' + 1);
        filename += 2;
    }

    expand_name(fcb, filename);
}

/*-----*/

expand_name(dest, src)
char    *src, *dest;
{
    /* Turn the filename at src into the name and type fields of an fcb
    * pointed to by dest (dest is the address of an fcb, src is an
    * ascii string).
    *
    *      "xxx.yyy"      becomes      "xxx      yyy"
    *      "xx*.x*"      becomes      "xx?????y??"
    *      "" or "*.*"    becomes      "???????????"
    *
    * The src parameter may be terminated by either a '\0' or a
    * space (' ').
    */
}
```



```

    */

register char    *dot, *end, i;

dot = ++dest;                                /* Skip past drive field */
for ( i=11 ; --i>=0 ; )                      /* Dest = ????????.??? */
    *dot++ = (*src) ? ' ' : '?';

dot = &dest[8];
end = &dest[10];

while( *src && *src != ' ' )
{
    if (*src == '*')
    {
        if ( dest < dot )
            while (dest < dot)
                *dest++ = '?';
        else
            while (dest <= end)
                *dest++ = '?';
    }
    else if (*src == '.')
        dest = dot;
    else
        *dest++ = *src;

    src++;
}
}

/*-----*/

```

```

#define DEF_FCB          0x5c                /* CP/M Default FCB location */
#define srchfirst(fcb)   ( bdos( 0x11, (fcb) ) & 0xff )
#define srchnext(fcb)    ( bdos( 0x12, (fcb) ) & 0xff )

```

```

int    do_wild(fname, argv, maxarg )
char   *fname, **argv ;
int    maxarg ;
{
    register char        *dest, *src;
    register unsigned int tmp;
    register int         i, argc = 0;
    extern  char         *alloc();

    /*      Expand fname (which may contain wild card characters)
     *      into a series of entries in argv. Return an updated argc.
     *      Maxarg is the maximum permitted value of argc.
     *      Fname may be terminated by either a '\0' or a ' '.
     */

    initfcb(fname, DEF_FCB);

    if ( (tmp = srchfirst(DEF_FCB)) != 0xff )
    {
        do
        {
            if( (dest = fname = alloc(13)) == 0 )
            {

```

(Continued on next page)

**ACCOUNTANTS TO ZOO KEEPERS
NEED DATABASE APPLICATIONS. HOW CAN
I SPEAK THEIR LANGUAGE?**



HERE'S HOW:

dBASE III™ SOFTWARE

Whether you're keeping track of accruals or zebras, dBASE III™ will easily create specialized applications which speak your customers' language.

You can quickly write applications for your customers because dBASE III contains a high-level language of plain-English commands that's powerful and easy to use. And, because we understand that your customers' needs often change, we made it easy to modify the structure of your database.

Let's say you've set up a database application for a zoo keeper. After using the application for six months, he discovers that he needs to specify the sex of all his animals, an item he forgot to ask

for when you created the application. You've got 1,397 records entered, but you have to change the structure of the database.

These commands:

USE ANIMALS






Selects the database.

MODIFY STRUCTURE

Move the cursor to the place where you want to insert the new field.



  Adds the new field.


The new field will be inserted above the cursor position.

Specifies that the new field is titled "SEX", has the de-

fault value of a character field, and is one column wide.

  ends your input.

 confirms that you're satisfied with what you've done. Now, your customer is ready to identify his zebras as boys or girls using the program you created for him.

dBASE III makes your programming effort quick and productive because it's an English-language exercise whether your customer is keeping track of cash flow or feeding schedules.

For a dealer near you call (800) 437-4329, ext. 2333. In Colorado (303) 799-4900, ext. 2333.



dBASE III and Ashton-Tate are trademarks of Ashton-Tate.
©Ashton-Tate 1985. All rights reserved.

Software from
ASHTON-TATE™
We'll put you in control.


```

                                perr("Not enough memory to expand wildcard");
                                exit(1);
        }

        src = ((char *)((tmp<<5) + 0x80)) + 1;

        if ( *(fname + 1) == ':' )
        {
                *dest++ = *fname;
                *dest++ = ':' ;
        }

        for( i = 1 ; i <= 11 ; i++ )
        {
                /*      Transfer name from DMA buffer to
                *      argument.
                */

                if( (*src & 0x7f) != ' ' )
                        *dest++ = *src++;
                else
                        src++;
                if( i == 8 )
                {
                        if ( (*src & 0x7f) == ' ' )
                                break;
                        else
                                *dest++ = '.' ;
                }
        }

        *dest = '\\0';
        argv[argc++] = fname;

        tmp = srchnext(DEF_FCB) ;

    }
    while( (tmp != 0xff) && (argc < maxarg) );
}

return( argc );
}

/*-----*/
haswild(str)
char *str;
{
    /*      Return true if str has a '*' or a '?' in it.
    */

    register int    c;

    do {
            c = *str++;
            if(c == '*' || c == '?')
                    return(1);
    } while(c && c != ' ');

    return(0);
}

```

End Listing One

Listing Two

LOADER.ASM Assembly language support routines for pipes

This loader is essentially that used by BDS C (written by Leor Zolman). The loader is copied to just below the tpa (into the default disk buffer) by loadldr. This location limits the size of the command tail for the next program in the pipe to $128 - 49 = 79$ characters (128 = buffer size, 49 = loader size). We're putting the loader where we do because an 8080 doesn't have a relative jump instruction. Since we are actually copying the loader to the place where it will execute, any labels defined at link time will be wrong (because they'll be pointing into the original code, not the copy). So, we have to copy it to some known place to know where to jump to when we loop. The bottom of the bdos, however, is not a known place. It will vary from system to system.

Before this loader is called, the calling program should put the new command line into the buffer at 0x80. It should also have an opened the file to be loaded with the fcb for that file in the default FCB space (at 0x5c).

```
closec equ 16      ; CP/M function: Close file.
reads  equ 20      ; CP/M function: Read sequential
sdma   equ 26      ; CP/M function: Set DMA address
```

```
fcbl   equ 5CH     ; Default FCB location
tpa     equ 100H    ; Start of transient program area
tbuff   equ 80H     ; start of command tail buffer
bdos     equ 05H    ; bdos jump vector
bdosv    equ 06H    ; pointer to bdos start address
```

```
-----
;                                LOADER                                (local)
;
; On entry:
;
;   D/E = the DMA address  ( 0x100 initially )
;   B/C = points at an fcb for the opened file. This fcb is physically
;         located just beneath the loader in memory.
;   SP  = just below the bdos.
;
; On exit:
;
;   All registers are destroyed
;
loader: lxi    d, tpa    ; destination address of new program
loadl:  push   d         ; push dma addr
        push   b         ; push fcb pointer
        mvi    c, sdma   ; set DMA address for new sector
        call   bdos
        pop    d         ; get pointer to working fcb in DE
        push   d         ; and re-push it
        mvi    c, reads  ; read a sector
        call   bdos
        pop    b         ; restore fcb pointer into BC
        pop    d         ; and dma address into DE
        ora    a         ; end of file?
        jz     tpa-8     ; if not, get next sector (goto 'load2:')
;
;-----
```

(Continued on next page)

C Chest (Listing Continued, text begins on page 10)

Listing Two

```

        mov     d,b           ;move the fcb pointer to d/e
        mov     e,c
        mvi     c,closec ; close the file
        call    bdos

        mvi     c,sdma ; reset DMA pointer, tbuf is at 0x80
        lxi     d,tbuff
        call    bdos
        jmp     tpa           ; and go invoke the program

load2:   lxi     h,80h        ;bump dma address
        dad     d
        xchg
        jmp     tpa-46        ;and go loop (at load1)

;-----
;                                LOADLDR_                (public)
;
;   This routine loads the loader into memory, modifies the stack pointer
;   and registers as required by the loader, and passes control to the
;   loader. Since loadldr jumps directly to the loader, it will not
;   return to the calling routine.
;
; On exit;
;   B = default FCB address      (0x5c)
;   D = tpa start address        (0x100)
;   SP = first address of BDOS

        public  loadldr_

loadldr_:
        lxi     d,loader      ; d = src;
        lxi     h,tpa-49      ; h = dest;
        mvi     b,49          ; b = 49;          /* size of loader */

L1:      ldax    d              ; do{
        mov     m,a            ;          *d++ = *h++;
        inx     d              ;
        inx     h              ; }
        dcr     b              ; while( --b > 0 );
        jnz     L1

        lxi     b,fcbl        ; b = address of fcb;
        lxi     d,tpa          ; d = dma address;

        lhld    bdosv         ; h = address of BDOS
        sphl                    ; sp = address of BDOS

        jmp     tpa-49        ; jump to the loader

end

```

End Listings

Once you choose Lattice,
our friends will C you through...

LATTICE INC.: LATTICE WINDOWS,
CURSES UNIX SCREEN CONTROL LIBRARY,
C-FOOD SMORGASBORD, dB-C ISAM
COMPATIBLE WITH dBASE II AND
III...

LIFEBOAT ASSOCI-

ATES: FLOAT 87 8087 SUPPORT
PACKAGE, HALO GRAPHICS
PACKAGE, PANEL SCREEN LI-
BRARY...

GREENLEAF SOFT-

WARE: THE GREENLEAF C
FUNCTIONS... **C SOURCE:**
BASIC C C FUNCTIONS FOR BA-
SIC USER... **SOFTCRAFT:**

BTRIEVE ISAM FILE SYSTEM,
BTRIEVE ISAM NETWORK FILE
SYSTEM...

BLAISE COMPUT-

ING: TOOLS, TOOLS2, VIEW
MANAGER, SCREEN, PACK-
AGE...

MORNING STAR

SYSTEMS: PROLIBRARY, PRO-

SCREEN... **CREATIVE SOLUTIONS:**

WINDOWS FOR C... **NOVUM**

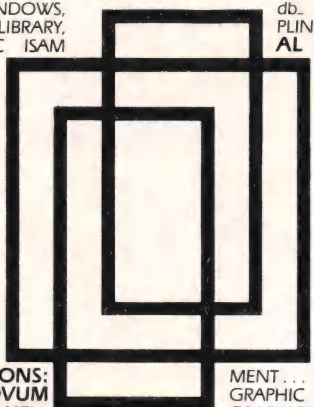
ORGANUM: C POWERS PACKS, MATH-

EMATICS POWER PACKS, ADVANCED POWER

PACKS, DATABASE POWER PACKS, TELE-

COMMUNICATIONS POWER PACKS W/
SOURCE... **PHACT ASSOCIATES:** PHACT

ISAM LIBRARY... **RAIMA CORPORATION:**



db_ VISTA DBMS... **PHOENIX:**
PLINK86, PFIX86... **RELATION-**
AL DATABASE SYSTEMS: C-

ISAM FILE ACCESS METH-

OD... **MINDBANK:** V-FILE

VIRTUAL MEMORY/FILE SYS-

TEM... **HUNTER &**

READY: VRTX C INTERFACE

LIBRARY... **GRAPHIC**

SOFTWARE SYSTEMS:

GSS DRIVERS, GSS TOOLKIT

KERNEL SYSTEM... **OPT-**

TECH DATA PROCESS-

ING: OPT-TECH SORT...

ACCUDATA SOFTWARE:

C-TREE ISAM, C-SORT

TRIO SYSTEMS:

C-INDEX + ISAM...

COMPU CRAFT: C VIEW

FORMS/WINDOW MANAGE-

SCIENTIFIC ENDEAVORS:

GRAPHIC PRESENTATION SCIENTIFIC

GRAPHICS... **LEMMA SYSTEMS,**

INC.: C LIBRARY... **ESSENTIAL SOFTWARE,**

INC.: C UTILITY LIBRARY... **SOFTWARE**

LABS: C UTILITIES PACKAGE... **FAIRCOM:** C-

tree BY FAIRCOM ISAM WITH SOURCE

Contact Lattice to learn how we can help your C program development.



LATTICE

P.O. Box 3072
Glen Ellyn, IL 60138
312/858-7950
TWX 910-291-2190

Circle no. 58 on reader service card.

LET db VISTA Manage Your Data

CALL
TOLL-FREE
1-800-843-3313
AT THE TONE 700-992

**The First DBMS Designed Exclusively
for C Language Applications Development**

Faster and better than ISAMs, file handlers, even end-user
DBMS's. POWER like a mainframe DBMS, PRICE like a
microcomputer utility, PORTABILITY like only C provides.

- * Written in C, under Unix.
- * Minimal data redundancy using the network database model.
- * Virtual memory disk accessing.
- * Fast B+-tree indexing method for key files.
- * Multiple key records - any or all data fields may be keys.
- * Three month extended applications support included.
- * PC-Write word processor/text editor included at no charge.

NOW INCLUDES SOURCE CODE AND FULLY PAID
RUN-TIME LICENSE.

30-DAY MONEY-BACK GUARANTEE

db_ VISTA versions

MS-DOS:	Lattice	\$495
	DeSmet	495
	Computer Innovations	495
	Aztec	495
	Mark Williams	495
UNIX:	Fortune 32-16	495
	Altos 586	495
	HP 9000/200 & 500	495
	AT&T 3B2	495
	Charles River	495
	Masscomp	495

db_ VISTA Manual 15

Development Packages

Lattice C w db_ VISTA	\$795
Lattice C only	395
DeSmet C w db_ VISTA	595

RAIMA
CORPORATION

11717 Rainier Avenue South
Seattle, WA 98178
206/772-1515



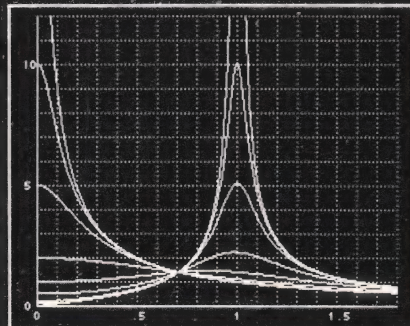
Only
\$295
without source code
\$395 with source code
until April 15, 1985

Circle no. 83 on reader service card.

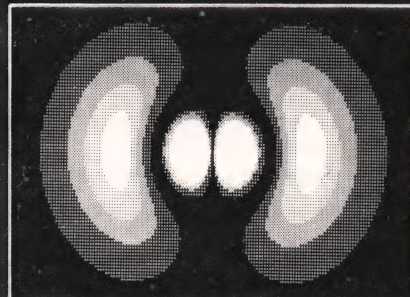
isys FORTH

for the Apple®][

Fixed point speed can rival that of float-
ing point hardware. But the details have
been a well kept secret—until now. The
following graphs were generated by fixed
point examples from the ISYS FORTH
manual.



Parallel Resonance with Damping
BASIC 213 sec ISYS FORTH 27 sec



Hydrogen 3p Orbital Cross-section
BASIC 492 sec ISYS FORTH 39 sec

- Fast native code compilation. Sieve benchmark: 33 sec
- Floating Point—single precision with transcendental
- Graphics—turtle & cartesian with 70-column character set
- Double Precision including D*/
- DOS 3.3 Files read & written
- FORTH-83 with standard blocks
- Full-Screen Editor
- Formatter for word processing
- Macro Assembler
- Price: \$99, no extra charges

ILLYES SYSTEMS

PO Box 2516, Sta A
Champaign, IL 61820

Technical Information:
217/359-6039, mornings

For any Apple][model, 48K or larger.
Apple is a registered trademark of Apple
Computer.

Circle no. 48 on reader service card.

REALIZABLE FANTASIES

The GNU Manifesto

by Richard Stallman

This month's installment of Realizable Fantasies features a guest appearance by Richard Stallman. Stallman, inventor of the original much-imitated Emacs editor and formerly of the Artificial Intelligence Lab at MIT, has worked extensively on compilers, editors, debuggers, command interpreters, the Incompatible Timesharing System (ITS) and the Lisp Machine operating system. He pioneered terminal-independent display support in ITS. Since then he has implemented one crashproof file system and two window systems for Lisp machines, and designed a third window system now being implemented; this one will be ported to many systems including GNU.—M.S.

GNU, which stands for Gnu's Not Unix, is the name for the complete Unix-compatible software system that I am writing so that I can give it away free to everyone who can use it. Many other programmers are helping me. Contributions of time, money, programs and equipment are greatly needed.

So far we have a portable C and Pascal compiler which compiles for Vax and 68000, an Emacs-like text editor with Lisp for writing editor commands, a yacc-compatible parser generator, a linker, and around 35 utilities. A shell (command interpreter) is nearly completed. When the kernel and a debugger are written, by the end of 1985 I hope, it will be possible to distribute a GNU system suitable for program development. After this we will add a text formatter, an Empire game, a spreadsheet, and hundreds of other things, plus on-line documentation. We hope to supply, eventually, everything useful that normally comes with a Unix system, and more.

GNU will be able to run Unix programs, but will not be identical with

Unix. We will make all improvements that are convenient, based on our experience with other operating systems. In particular, we plan to have longer filenames, file version numbers, a crash-proof file system, filename completion, perhaps, terminal-independent display support, and eventually, a Lisp-based window system through which several Lisp programs and ordinary Unix programs can share a screen.

Both C and Lisp will be available as system programming languages. We will try to support UUCP, MIT Chaonet, and Internet protocols for communication.

GNU is aimed initially at machines in the 68000/16000 class, with virtual memory, because they are the easiest machines to make it run on. The extra effort to make it run on less powerful machines will be left to someone who wants to use it on them.

Why I Must Write GNU

If I like a program, I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I cannot in good conscience sign a nondisclosure or software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies. My efforts were wasted. I cannot remain in an institution where such things are done for me against my will.

So that I can continue to use computers without violating my principles I have decided to put together a body of free software sufficient to enable me to get along without any software that is not free. I have resigned from the AI lab to deny MIT any legal excuse for preventing me from giving GNU away.

Why GNU Will Be Compatible with Unix

Unix is not my ideal system, but it is

not too bad. The essential features of Unix seem to be good ones, and I think I can fill in what Unix lacks without spoiling them. Furthermore a system compatible with Unix would be convenient for many other people to adopt.

How GNU Will Be Available

GNU is not in the public domain. Everyone will be permitted to modify and redistribute GNU, but no distributor will be allowed to restrict its further redistribution. That is to say, proprietary modifications will not be allowed. I want to make sure that all versions of GNU remain free.

Why Many Other Programmers Want to Help

I have found many other programmers who are excited about GNU and want to help. Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires that they feel like competitors with other programmers rather than like comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now in use essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money.

By working on and using GNU rather than proprietary programs, we can be hospitable to everyone and obey the law. In addition, GNU serves as an example to inspire and a banner to rally others to join us in sharing. This can give us a feeling of harmony, which is impossible if we use software that is

not free. For about half the programmers I talk to, this is an important happiness that money cannot replace.

How You Can Contribute

I am asking computer manufacturers for donations of machines and money. I'm asking individuals for donations of programs and work.

One computer manufacturer has already offered to provide a machine. We can use more. One consequence you can expect if you donate machines is that GNU will run on them at an early date. The machine should be able to operate in a residential area, and not require sophisticated cooling or power.

I have found very many programmers eager to contribute part-time work to GNU. For most projects, such part-time distributed work would be very hard to coordinate; the parts, written independently, would not work together. But for the particular task of replacing Unix, this problem is absent. A complete Unix system contains hundreds of utility programs, each of which is documented separately. Most interface specifications are fixed by Unix compatibility. If each contributor can write a compatible replacement for a single Unix utility, and make it work properly in place of the original on a Unix system, then these utilities will work right when put together. Even if Murphy creates a few unexpected problems, assembling these components will be a feasible task. (The kernel will require closer communication and will be worked on by a small, tight group.)

If I get donations of money, I may be able to hire a few people full or part-time. The salary won't be high by programmer's standards, but I'm looking for people for whom building community spirit is as important as making money. I view this as a way of enabling dedicated people to devote their full energies to working on GNU by sparing them the need to make a living in another way.

Why All Computer Users Will Benefit

Once GNU is written, everyone will be able to obtain good system software free, just like air. This means much more than just saving everyone the price of a Unix license. It means that

much wasteful duplication of system programming will be avoided. This effort can go instead into advancing the state of the art.

Complete system sources will be available to everyone. As a result, a user who needs changes in the system will always be free to make them himself, or hire any available programmer or company to make them for him. Users will no longer be at the mercy of one programmer or company that owns the sources and is in sole position to make changes.

Schools will be able to provide a superior educational environment by encouraging all students to study and im-

prove the system code. Harvard's computer lab used to have the policy that no program could be installed on the system if its sources were not on public display, and upheld it by actually refusing to install certain programs. I was very much inspired by this.

Finally, the overhead of considering who owns the system software and what one is or is not entitled to do with it will be lifted. Arrangements to make people pay for using a program, including licensing of copies, always impose a tremendous cost on society through the cumbersome mechanisms necessary to figure out how much (that is, which programs) a person must pay



dBASE II® outfoxed!

Who says the most popular database management system is the best?

Introducing **FoxBASE II™**, the new relational database management system that's dBASE II source compatible. It does everything dBASE II does ... plus a whole lot more.

- Runs 3 to 5 times faster
- Sorts up to 20 times faster
- Permits up to 48 fields per record ... 50% more than dBASE II
- Supports full type-ahead
- Compiles program sources into compact object code
- Comes with a sophisticated online manual and HELP facility
- Has twice as many memory variables

What's more, it costs less. **MS-DOS \$395. AOS/VS \$995.**

FoxBASE II is available now for: IBM-PC, IBM-PC/XT, COMPAQ & IBM compatibles, TI Professional, DG Desktop, DG MV Series, and many others. Call or write today for more information.

Developed by

DACOR

COMPUTER SYSTEMS

dBASE II is a registered trademark of Ashton-Tate
FoxBASE II is a trademark of Fox Software Inc.

FoxBASE II
from FOX SOFTWARE INC.

13330 Bishop Road, P.O. Box 269, Bowling Green, OH 43402 / 419-354-3981 / TWX 810-499-2989

Circle no. 40 on reader service card.

for. Furthermore, only a police state can force everyone to obey. Consider the analogy of a space station where air must be manufactured at great cost: charging each breather per liter of air might be fair, but wearing the metered oxygen mask all day and all night would be intolerable even if everyone could afford to pay the bill. And the TV cameras everywhere to see if you ever took it off would be outrageous. It would be better to support the air plant with a head tax and chuck the masks. Copying all or parts of a program is as natural to a programmer as

breathing, and as productive. It ought to be as free.

Some easily rebutted objections to GNU's goals

"Nobody will use it if it is free, because that means they can't rely on any support. You have to charge for the program to pay for providing the support." If people would rather pay for GNU plus service than get GNU free without service, a company to provide just service to people who have obtained GNU free ought to be profitable.

We must distinguish between support in the form of real programming work and mere handholding. The former is something one cannot rely on from a software vendor. If your problem is not shared by enough people, the vendor will tell you to get lost. If your business needs to be able to rely on support, the only way is to have all the necessary sources and tools. Then you can hire any available person to fix your problem and you will not be at the mercy of any individual. With Unix, the price of sources puts this out of consideration for most businesses. With GNU this will be easy. It is still possible that there will be no available competent person, but this problem cannot be blamed on distribution arrangements. GNU does not eliminate all the world's problems, only some of them.

Meanwhile, the users who know nothing about computers need handholding, i.e., they need for *others* to do for them the things which they could easily do *themselves*, but don't know how to. Such services could be provided by companies that sell just handholding and repair service. If it is true that users would rather spend money and get a product with services, they will also be willing to buy the service, having got the product free. The service companies will compete in quality and price; users will not be tied to any particular one. Meanwhile, those of us who don't need the service should be able to use the program without paying for the service.

"You cannot reach many people without advertising, and you must charge for the program to support that. It's no use advertising a program people can get free." There are various forms of free or very cheap publicity that can be used to inform numbers of computer users about something like GNU. But it may be true that one can reach more microcomputer users with advertising. If this is really so, a business which advertises the service of copying and mailing GNU for a fee ought to be successful enough to pay for its advertising and more. This way, only the users who benefit from the advertising pay for it. On the other hand, if many people get GNU from their friends, and such companies don't succeed, this will show that advertising was not really necessary to spread GNU. Why is it that free market advo-

LPA PROLOG Language Interpreter License - MS-DOS	\$295.00
LPA Sigma-PROLOG UNIX Version — 68,000	\$695.00
LPA Sigma-PROLOG UNIX Version — VAX	\$1500.00
LPA PROLOG Language Interpreter Documentation	45.00
APES Augmented PROLOG — MS-DOS	\$295.00
APES for UNIX Systems	\$695.00/\$1500.00
APES Documentation	10.00
Pkg. LPA PROLOG & APES — MS-DOS	\$495.00

PROGRAMMING LOGIC SYSTEMS
 31 Crescent Drive
 Milford, Connecticut 06460
 (203) 877-7988

CP/M is a trademark of Digital Research Inc.
 MS-DOS is a registered trademark of Microsoft Corp.
 PC-DOS is a registered trademark of IBM
 UNIX is a trademark of Bell Labs.

Educational & Pkge.
 Discounts available
 Prices subject to change

Circle no. 87 on reader service card.

cates don't want to let the free market decide this?

"My company needs a proprietary operating system to get a competitive edge." GNU will remove operating system software from the realm of competition. You will not be able to get an edge in this area, but neither will your competitors be able to get an edge over you. You and they will compete in other areas, while benefiting mutually in this one. If your business is selling an operating system, you will not like GNU, but that's tough on you. If your business is something else, GNU can save you from being pushed into the expensive business of selling operating systems. I would like to see GNU development supported by gifts from many manufacturers and users, reducing the cost to each.

"Don't programmers deserve a reward for their creativity?" If anything deserves a reward, it is social contribution. Creativity can be a social contribution, but only insofar as society is free to use the results. If programmers deserve to be rewarded for creating innovative programs, by the same token they deserve to be punished if they restrict the use of these programs.

"Shouldn't a programmer be able to ask for a reward for his creativity?" There is nothing wrong with wanting pay for work, or seeking to maximize one's income, as long as one does not use means that are destructive. But the means customarily used in the area of software development today are based on destruction. Extracting money from users of a program by restricting their use of it is destructive because the restrictions reduce the amount that and the ways in which the program can be used. This reduces the amount of wealth that humanity derives from the program. When there is a deliberate choice to restrict, the harmful consequences are deliberate destruction. The reason a good citizen does not use such destructive means to become wealthier is because, if everyone did so, we would all become poorer from the mutual destructiveness. This is Kantian ethics, or, the Golden Rule. Since I do not like the consequences that result if everyone hoards information, I am required to consider it wrong for one person to do so. Specifically, the desire to be rewarded for one's creativity does

not justify depriving the world in general of all or part of that creativity.

"Won't programmers starve?" I could answer that nobody is forced to be a programmer. Most of us cannot manage to get any money for standing on the street and making faces. But we are not, as a result, condemned to spend our lives standing on the street making faces, and starving. We do something else. But that is the wrong answer, because it accepts the questioner's implicit assumption that without ownership of software, programmers cannot possibly be paid a cent. Supposedly it is all or nothing. The real reason programmers will not starve is because it will still be possible for them to get paid for programming: just not as much as now.

Restricting copying is not the only means for making a profit in software development. It is the most common means because it brings in the most money. If it were prohibited, or rejected by the customer, software business would move to other methods of profit-making that are now used less often. Probably programming would not be as lucrative as it now. But that is not an argument against the change. It is not considered an injustice that sales clerks make the salaries that they now do. If programmers made the same, that would not be an injustice either. (In practice they would still make considerably more than that.)

"Don't people have a right to control how their creativity is used?" Control over the use of one's ideas really constitutes control over other people's lives; and it is usually used to make their lives more difficult. People who have carefully studied the issue of intellectual property rights (such as lawyers) say that there is no intrinsic right to intellectual property. The kinds of supposed intellectual property rights that the government recognizes were created by specific acts of legislation for specific purposes. For example, the patent system was established to encourage inventors to disclose the details of their inventions. Its purpose was to help society rather than to help inventors. At the time, the life span of 17 years for a patent was short compared with the rate of advance of the state of the art. Since patents are an issue only among manufacturers, for

AT LAST

S-100 ↔ 488

THAT

DOES

EVERYTHING

YOU WANT

IT TO DO



D&W DIGITAL, INC.
20655 Hathaway Avenue
Hayward, California 94541
(415) 887-5711

whom the cost and effort of a license agreement are small compared with setting up production, the patents often do not do much harm. They do not obstruct most individuals who use patented products.

The idea of copyright did not exist in ancient times, when authors frequently copied lengthy extracts from other authors in works of non-fiction. This practice was useful, and is the only way many authors' works have survived even in part. The copyright system was created expressly for the purpose of encouraging authorship. In the domain for which it was invented—books, which could be copied economically only on a printing press—it did little harm, and did not obstruct most of the individuals who read the books.

All intellectual property rights are just licenses granted by society because it was thought, rightly or wrongly, that society as a whole would benefit by granting them. But in any particular situation, we have to ask: Are we really better off granting such license? What kind of act are we licensing a person to do? The case of programs today is very different from that of books a hundred years ago. The fact that the easiest way to copy a program is from one neighbor to another, the fact that a program has both source code and object code, which are distinct, and the fact that a program is used rather than read and enjoyed, combine to create a situation in which a person who enforces a copyright is harming society as a whole both materially and spiritually; in which a person should not do so regardless of whether the law enables him to or not.

"Won't everyone stop programming without a monetary incentive? Actually, many people will program with absolutely no monetary incentive. Programming has an irresistible fascination for some people, usually the people who are best at it. There is no shortage of professional musicians who keep at it even though they have no hope of making a living that way. But really this question, though commonly asked, is not appropriate to the situation. Pay for programmers will not disappear, only become less. So the right question is: Will anyone program with a reduced monetary incentive? My experience shows that they will.

For more than ten years, many of the world's best programmers worked at the Artificial Intelligence Lab for far less money than they could have had anywhere else. They got many kinds of non-monetary rewards: fame and appreciation, for example. And creativity is also fun, a reward in itself. Then most of them left when offered a chance to do the same interesting work for a lot of money. What the facts show is that people will program for reasons other than riches; but if given a chance to make a lot of money as well, they will come to expect and demand it. Low-paying organizations do poorly in competition with high-paying ones, but they do not have to do badly if the high-paying ones are gone.

"We need the programmers desperately. If they demand that we stop helping our neighbors, we have to obey." You're never so desperate that you have to obey this sort of demand. Remember, millions for defense, but not a cent for tribute.

"Programmers need to make a living somehow." There are plenty of ways by which programmers can make a living without selling the right to use a program. Here are a number of examples:

- A manufacturer introducing a new computer will pay for the porting of operating systems onto the new hardware.
- The sale of teaching, handholding and maintenance services could also employ programmers.
- People with new ideas could distribute programs as freeware, asking for donations from satisfied users. I am told that several people are already working this way successfully.
- Users with related needs can form users' groups, and pay dues. A group would contract with programming companies to write programs that the group's members would like to use.

All sorts of development can be funded with a software tax:

- Suppose everyone who buys a computer has to pay x percent of the price as a software tax. The government gives this to an agency like the NSF to spend on software development.
- But if the computer buyer makes a donation to software development

himself, he can take a credit against the tax. He can donate to the project of his own choosing—often, chosen because he hopes to use the results when it is done. He can take a credit for any amount of donation up to the total tax he had to pay.

- The total tax rate could be decided by vote of the payers of the tax, weighted according to how much tax they paid in the previous year.

The consequences:

- The computer-using community supports software development.
- This community decides what level of support is needed.
- Users who care which projects their share is spent on can choose this for themselves.

In the long run, making programs free is a step toward the post-scarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming.

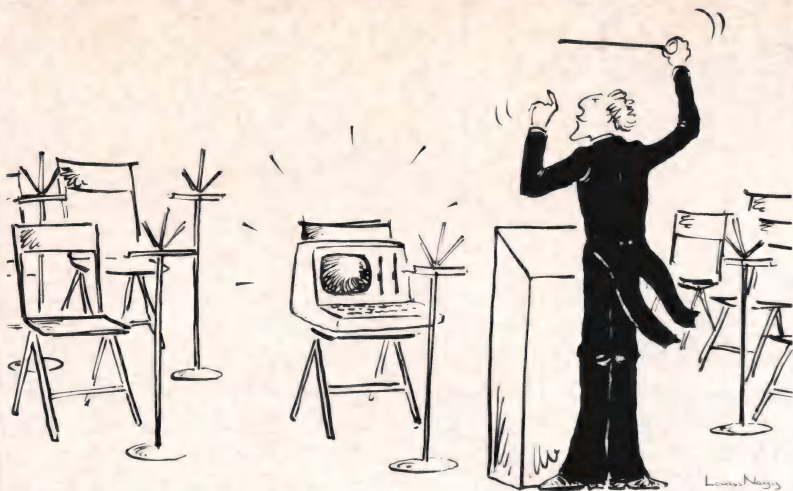
We have already greatly reduced the amount of work that the whole society must do for its actual productivity, but only a little of this has translated itself into leisure for workers because much nonproductive activity is required to accompany productive activity. The main causes of this are bureaucracy and isometric struggles against competition. Free software will greatly reduce these drains in the area of software production. We must do this in order for technical gains in productivity to translate into less work for us.

Richard Stallman, 166 Prospect Street, Cambridge, MA 02139. Copyright © 1985 Richard Stallman. Permission is granted to make and distribute copies of this article as long as the copyright and this notice appear, and the copies are distributed at no charge.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 191.



Would you hire an entire band when all you need is one instrument? Of course not.

So why use a whole orchestra of computers when all you need is one to develop software for virtually any type of micro-processor?

The secret? Avocet's family of cross-assemblers. With Avocet cross-assemblers you can develop software for practically every kind of processor — *without having to switch to another development system along the way!*

Cross-Assemblers to Beat the Band!

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 4 years of actual use. Ask NASA, IBM, Xerox or the hundreds of other organizations that use them. Every time you see a new micro-processor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on almost any personal computer and process assembly language for the most popular microprocessor families.

Your Computer Can Be A Complete Development System

Avocet has the tools you need to enter and assemble your software and finally cast it in EPROM:

VEDIT Text Editor makes source code entry a snap. Full-screen editing plus a TECO-like command mode for advanced tasks. Easy installation - INSTALL program supports over 40 terminals and personal computers. Customizable keyboard layout. CP/M-80, CP/M-86, MSDOS, PC DOS.....\$150

EPROM Programmers let you program, verify, compare, read, display EPROMS but cost less because they communicate through your personal computer or terminal. No personality modules! On-board intelligence provides menu-based setup for 34 different EPROMS, EEPROMS and MPUs (40-pin devices require socket adaptors). Self-contained unit with internal power supply, RS-232 interface, Textool ZIF socket. Driver software (sold separately) gives you access to all programmer features through your computer, lets you download cross-assembler output files, copy EPROM to disk.

Model 7228 Advanced Programmer — Supports all PROM types listed. Superfast "adaptive" programming algorithm programs 2764 in 1.1 minutes.

Model 7128 Standard Programmer — Lower-cost version of 7228. Supports all PROM types except "A" versions of 2764 and 27128. Standard programming algorithm programs 2764 in 6.8 minutes.

Avocet Cross-assembler	Target Microprocessor	CP/M-80	CP/M-86 IBM PC, MSDOS**
XASM04 NEW	6804	\$ 250.00	\$ 250.00
XASM05	6805	200.00	250.00
XASM09	6809	200.00	250.00
XASM18	1802/1805	200.00	250.00
XASM48	8048/8041	200.00	250.00
XASM51	8051	200.00	250.00
XASM65	6502/65C02	200.00	250.00
XASM68	6800/01, 6301	200.00	250.00
XASM75	NEC 7500	500.00	500.00
XASM85	8085	250.00	250.00
XASM400	COP400	300.00	300.00
XASMF8	F8/3870	300.00	300.00
XASMZ8	Z8	200.00	250.00
XASMZ80	Z80	250.00	250.00
XMAC682 NEW	68200	595.00	595.00
XMAC68K NEW	68000/68010	595.00	595.00

Model 7956 and 7956-SA Gang Programmers — Similar features to 7228, but program as many as 8 EPROMS at once. 7956-SA stand-alone version copies from a master EPROM. 7956 lab version has all features of stand-alone plus RS-232 interface.

EPROM: 2758, 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 2508, 2516, 2532, 2564, 68764, 68766, 5133, 5143. **CMOS:** 27C16, 27C32, 27C64, MC6716. **EEPROM:** 5213, X2816A, 48016, I2816A, 5213H. **MPU (w/adaptor):** 8748, 8748H, 8749, 8749H, 8741, 8742, 8751, 8755.

7228	Advanced Programmer	\$ 549
7128	Standard Programmer	429
7956	Laboratory Gang Programmer	1099
7956-SA	Stand-Alone Gang Programmer	879
GDX	Driver Software	95
481	7448 Family Socket Adaptor	98
511	8751 Socket Adaptor	174
755	8755 Socket Adaptor	135
CABLE	RS-232 Cable (specify gender)	30

HEXTRAN Universal HEX File Converter — Convert assembler output to other formats for downloading to development systems and target boards. Also useful for examining object file, changing load addresses, extracting parts of files. Converts to and from Intel, Motorola, MOS, RCA, Fairchild, Tektronix, TI, Binary and HEX/ASCII Dump formats. For CP/M, CP/M-86, MSDOS, PC DOS.....\$250

Ask about UNIX.

68000 CROSS-ASSEMBLER — With exhaustive field testing completed, our 68000 assembler is available for immediate shipment. XMAC68K supports Motorola standard assembly language for the 68000 and 68010. Macros, cross-reference, structured assembly statements, instruction optimization and more. Linker and librarian included. Comprehensive, well-written manual.

To find out more, call us toll-free.

1-800-448-8500

(in the U.S. Except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available — please specify. Prices do not include shipping and handling — call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft



Sales and Development:
10 Summer Street
P.O. Box 490, Dept.385-DDJ
Rockport, Maine 04856
(207) 236-9055 Telex: 467210 AVOCET CI

Corporate Offices:
804 South State Street
Dover, Delaware 19901

Programming in Logic

by John Malpas

Logic programming has been relatively little known in the United States until recently. A number of factors are converging, however, that augur well for an increase in its use and popularity. First of all, several interpreters offering respectable performance and support are now commercially available, and apparently other, high-performance interpreters will soon be released. Second, the lukewarm attitude of hardware manufacturers towards logic programming seems to be changing: the new Tektronix machine was released with Prolog already running on it, and Apple plans to offer a version of Prolog for the Macintosh. Third, there is an increasing demand from computer people on all levels for artificial intelligence technology, and one of the most promising areas of AI research has been the area of programming in logic.

Logic programming (see Table 1,

“:-” is read as *if*, and the commas are read as *and*, so the whole clause can be read: *The conclusion is true if condition1 and condition2 and ... conditionN are all true.*

Prolog is the most widespread logic programming language. A Prolog interpreter is based on a general purpose inference mechanism (typically using the resolution and unification techniques); to take advantage of this mechanism, the programmer need only declare the relationships between data items in Horn clause form. A logic program does not need overt control constructs (such as *for* or *while*) because the interpreter makes most of the control decisions.

One consequence of this approach is portability. A logic program could be made to run on all of the common versions of Prolog with relative ease. Because logic programs depend on logic, not on the implementation details of

Prolog's admirers claim that to use the language is to program in logic. Is the claim true, and is programming in logic something we should want to do?

page 37) is based on the idea that statements in first-order predicate logic, cast in Horn clause form, can be used directly as a programming language. In Horn clause form, one conclusion is followed by zero or more conditions, as follows:

```
conclusion :-  
    condition1,  
    condition2,  
    ...,  
    conditionN.
```

John Malpas, Pulsetrain, 747 Greenwich St., New York, NY 10014. Copyright 1984, Pulsetrain.

any particular interpreter, they are guaranteed to be upwardly compatible with faster interpreters or more efficient hardware. [But see D. E. Cortesi's discussion of parallelism (page 50) for an important caveat.—Ed.] Because the interpreter makes most of the fundamental control decisions, and because of the portability over hardware, any developments in the implementation of Prolog interpreters and in the hardware substratum are particularly interesting. There have been several such developments recently.

The first Prolog was implemented by Colmerauer in Fortran and was very slow and awkward to use (for the

development of Prolog see Table 2, below right). In 1977 Warren and Pereira wrote an interpreter/compiler in DEC-10 assembly language that showed that Prolog could be just as efficient as Lisp. A number of versions have been written since then in C for Unix machines. Clark and McCabe's original Micro-PROLOG was in assembly language on PCs, but the latest version (called Sigma-PROLOG) is written in C and runs on larger machines. There are several versions of Prolog written in Lisp for Lisp machines.

In 1983, ICOT (where the Japanese fifth generation project is being developed) announced their Personal Sequential Inference Machine, which has a processor instruction set optimized to run Prolog—a Prolog machine. When I saw this computer, it had 80MB of main memory. Warren has also followed the approach of optimizing processor microcode to run Prolog. His company (Quintus Computer) has a fast Prolog running on a VAX and a Sun workstation. Other fast Prolog interpreters are under development at UC Berkeley and Argonne National Laboratories. Prolog for machines based on parallel architecture is being developed by Shapiro in Israel, at Imperial College in London, and at ICOT.

In the area of microcomputers, there are currently three commercial versions of Prolog for the IBM PC, and Apple will announce a version of Prolog for the Macintosh this spring. There are other implementations as well, some written in Lisp and of value for learning the language. The three versions for the PC are true commercial products (for sources of Prolog see Table 3, page 38).

Features of Prolog

The fact that Prolog is an implementation of logic as a programming language makes the language distinctive in several ways.

Prolog Can Express Knowledge

Logic is a language capable of expressing situations and problems that provides a way to describe accurately decision-making behavior. Logic has precise semantics; that is, it is possible to understand unambiguously the meaning of a logical expression. Logical expressions can be represented by

R.A. Kowalski, *Logic for Problem Solving*, North Holland, 1979 (Much of the logic programming research done in the last five years has been inspired by this book.)

K.L. Clark and F.G. McCabe, *Micro-PROLOG: Programming in Logic*, Prentice Hall, 1984 (This is the best introduction to programming in Prolog; all examples are in Micro-PROLOG syntax.)

W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, Springer-Verlag, 1981 (This is the standard introductory text for Prolog in DEC-10 syntax.)

L. Wos, R. Overbeek, E. Lusk, and J. Boyle, *Automated Reasoning, Introduction and Applications*, Prentice Hall, 1984 (This contains a good introduction to symbolic logic.)

P.J. Hayes, "In Defense of Logic," *Proceedings of 5th IJCAI*, 1977

Table 1
Further Reading

1879: Gotlob Frege invents predicate calculus.

1965: J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *JACM* [12(1) 1965] pp. 23–41.

1968: D.W. Loveland, "Mechanical Theorem Proving by Model Elimination," *JACM* [15(2) 1968].

1971: Robert Kowalski and Donald Kuehner, "Linear Selection with Resolution Function," *Artificial Intelligence* (2 1971) pp. 227–260.

1973: Alain Colmerauer at Marseilles writes first Prolog in Fortran (see Philippe Russel, *Technical Report of the University of Marseilles*, Sept. 1975).

1974: Robert Kowalski, "Predicate Logic as a Programming Language," *Proceedings of IFIP*, 1974.

1977: David Warren and Fernando Pereira's DEC-10 Prolog interpreter/compiler, University of Edinburgh (see Warren and Pereira, "Prolog: The Language and Its Implementation Compared to Lisp," *ACM SIGPLAN Notices* [12(8) 1977]).

1980: C-Prolog for Unix systems; Micro-PROLOG for PCs from Imperial College (see K.L. Clark and F.G. McCabe, *Micro-PROLOG: Programming in Logic* Prentice Hall, 1984).

1981: Fifth generation project announced by ICOT in Japan, based on logic programming technology (see *Fifth Generation Computer Systems*, ed. T. Moto-Oka, North Holland, 1982).

1982: LM-Prolog for Lisp machines (see K.M. Kahn and M. Carlsson, "LM-Prolog User Manual, Release 1.0," *Technical Report of Uppsala University*, July 1983).

1983: E.Y. Shapiro designs Concurrent Prolog at ICOT (see Shapiro, "A Subset of Concurrent Prolog and Its Interpreter," *Technical Report #3 ICOT*, Oct. 1983).

Table 2
Development of Logic Programming

different logic languages such as Horn clauses. An expression in one logic language is translatable into an expression in any other logic language.

Kowalski has stated that "algorithm = logic + control," which can be understood as follows: an algorithm is composed of the set of data relationships defining an application domain (that is, its logic) and the control information about how to use those data relationships. In logic programming, because the interpreter takes care of the control part of this equation, a programmer's primary responsibility is to learn the internal logic of an application domain. Writing an application program in Prolog amounts to expressing this internal logic as Horn clauses.

Prolog Describes, It Does Not Simulate

When a procedural language program is used to model a process in the world, the relationship between the program and the world is one of simulation. This means that the program is working on the same level as the process itself, and so must have large amounts of data *from* the process.

On the other hand, when a logic program is used to model a process in the world, the relationship between the program and the world being modeled is one of *description*. The program exists on a higher level than the process it describes, because it contains statements and generalizations (knowledge) *about* the process.

A program is a simulation of the world whenever the state of the computer itself (i.e., CPU registers, memory, etc.) is an integral part of the model. In the case of a descriptive program, however, the state of the computer functions only to implement the logic of the program and does not play a role in the model *per se*.

Prolog Does Not Use Destructive Assignment

Logic programming achieves its descriptive power by dispensing with the destructive assignment statement. Once a logical variable assumes a value, the programmer cannot arbitrarily decide to give it another value. In contrast, all procedural languages depend upon destructive assignment as their most essential semantic feature.

Destructive assignment is best understood as an *optimization of memory resources*. In a language based on destructive assignment, any change in a data value in the program is irrevocable; there will be no trace of previous values of the variable unless the programmer has made provisions to save them.

Prolog Statements Have Both Declarative and Procedural Interpretations

There are two ways to read a Horn clause logic program: declaratively and procedurally. Consider the example rule:

```
pays_too_much_rent (X) :-
    lives_in (X, new_york).
```

There is a procedural way to read this rule:

To find someone who pays too much rent,
find a resident of New York.

or

To prove that John pays too much rent,
verify that he lives in New York.

But it is also possible to read the rule declaratively, which enables Horn clauses to be used as a knowledge representation language:

Someone who pays too much rent is
someone who lives in New York.

or

All people who live in New York
pay too much rent.

In the declarative reading, the rule becomes a definition of a relationship

Programming Logic Systems, Inc.
31 Crescent Drive
Milford, CT 06460

Micro-PROLOG (IBM PC, other PCs, CP/M)
Sigma-PROLOG (written in C for Unix machines)
APES (A Prolog Expert System Shell)

Logicware
1000 Finch Avenue West
Toronto, Ontario
Canada M3J 2V5

M-Prolog (assembly language on IBM mainframe, cross compiled to PC, VAX, *et al.*)

Expert Systems International
1150 First Avenue
King of Prussia, PA 19406

PROLOG-1 (assembly language on MSDOS, CP/M, VAX, Macintosh)
ES/P Advisor (Expert System Shell)

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

C-Prolog (written in C for Unix machines with at least 256K addressable memory)

University of New Hampshire
UNH Prolog (written in C for Berkeley Unix)

University of Edinburgh
DEC-10 Prolog (assembly language)

Waterloo University, Canada
Waterloo Prolog (assembly language on IBM mainframes)

Public Domain
UNSW Prolog (written in C for Unix machines)

Table 3
Sources of Prolog

CP/M SYSTEM INVENTORY LIQUIDATION



Old operating systems never die, they get better with age! Multitech Electronics, a large manufacturer of complete computer systems has an inventory of CP/M computers that must go. These fully guaranteed high performance units are new factory packaged systems with the CP/M 2.2 operating system, 64K of RAM, dual floppy disk drives, and CBASIC programming language. You'll get more than what you paid for with complete systems at bargain prices.

FULL CP/M SYSTEM SPECS

	MIC-501	MIC-504
CPU	Z-80A	Z-80A
RAM MEMORY	64K	64K
STORAGE	500K BYTES	2M BYTES
SERIAL COMMUNICATIONS	DUAL RS-232C PORTS	DUAL RS-232C PORTS
PARALLEL COMMUNICATIONS	SINGLE PARALLEL PORT	SINGLE PARALLEL PORT
PRICE	\$500	\$700

This low price also includes: full compatibility with the wide bank of CP/M software, CP/M 2.2 operating system with utilities and manuals, CBASIC 2 programming language, as well as a User's Manual and Service Manual. In addition you'll get a 30 day full warranty. And Multitech will also throw in a Liberty Freedom 100 terminal for \$400.00 with the purchase of an MIC-500

system. Call Multitech today for more on your CP/M solutions. Cash or credit cards only. Limited quantities available.



Multitech ELECTRONICS
INC.

195 WEST EL CAMINO REAL
SUNNYVALE, CALIFORNIA 94087
TELEX: 755042 MAC SUVL
IN CALIFORNIA, CALL: (408) 773-8400
OUTSIDE CALIFORNIA, CALL: (800) 538-1542

To give some idea of what logic programming code is like, it will be useful to examine a simple program. The simplest Horn clause is an assertion of fact, consisting of a conclusion followed by no conditions:

`lives_in(john, new_york).`

which can be read: *John lives in New York*. A rule is another form of Horn clause that shows the dependence of one fact on other facts:

`pays_too_much_rent(john) :-`

`lives_in(john, new_york).`

which can be read: *If John lives in New York, he pays too much rent*. The word "john" is a constant, and these rules and facts contain explicit knowledge applicable to "john."

A logical variable in Prolog is written as a word beginning with a capital letter. The "too much rent" rule above can be generalized so that it applies to all New York residents by replacing "john" with a variable:

`pays_too_much_rent(X) :-
lives_in(X, new_york).`

which may be read: *Anyone who lives in New York pays too much rent*. The variable "X" will match any constant, so this rule can be used to assert that the rent of any New York resident is excessive.

A rule utilizing variables in this way contains implicit knowledge that is applicable to all people who live in New York.

Prolog is built on the model of symbolic logic created by Frege. There are two varieties of logic to consider: propositional logic and first-order predicate logic.

Propositional logic considers only relations between (unanalyzed) propositions, and only such relations as bear on the propositions' truth or falsity. Propositions can be combined with other propositions to form new propositions via the logical connectives **and**, **or**, **not**, and **implies**. Propositional logic concerns itself with what propositions can be derived from what other propositions. For example, from proposition **P or Q**, where **P** and **Q** are propositions, we can derive **Q**. Also, from **P implies Q** and **P** we can derive **Q**.

First-order predicate logic involves more complex expressions (formulae) that contain variables and that become propositions only when values from some domain of discourse are substituted for these variables. Thus, **P(x)** becomes a proposition and can be true or false only when the variable **x** is assigned a specific value or is otherwise quantified. The ways of quantifying **x** that are particularly important for first-order predicate logic are via the universal and existential quantifiers, which turn the variable **x** into, respectively, **for every x** and **there exists an x**. First-order predicate logic is sufficiently powerful to express a good deal of mathematics and to serve as the model for a programming language. Prolog depends on the resolution technique in its implementation of first-order predicate logic.

Resolution is a decision algorithm for satisfiability in first-order predicate logic. Because the truth of a for-

mula in first-order predicate logic depends on how values are assigned to its variables, one generally talks about the satisfiability of a formula, rather than about its truth. A formula is satisfiable if and only if it is true under some admissible assignment of values to its variables. Thus, if **P(a)** means "**a** is a prime number" and **Q(a)** means "**a** is greater than 11 and less than 17," the formula **there exists an x such that (P(x) and Q(x))** is satisfiable, namely, by the value assignment **x = 13**.

Resolution is a partial decision algorithm; it will invariably determine the unsatisfiability of an unsatisfiable formula, though it is not guaranteed to halt if the formula is, in fact, satisfiable. (This halting problem is a consequence of the fundamental unsolvability of first-order predicate logic, not a deficiency of the algorithm.) By recasting the formula, though, you can demonstrate its satisfiability (if it *is* satisfiable), throwing the halting problem into the unsatisfiability-proving half of the task.

The resolution method requires that the formula be in a particular form, variably referred to as clause form, Horn clause form, or conjunctive normal form. (These terms are not really interchangeable, but for this sketchy discussion we can regard them as so.) The basic idea is this: we eliminate all existential quantifiers by a trick that involves treating the variables they quantify as functions of a particular sort. Once the existential quantifiers are eliminated, we simply drop universal quantifiers, assuming that all remaining variables are universally quantified. And we transform the formula into a logically equivalent formula whose top-level components are all

connected by **ands**, whose second-level components are all connected by **ors**, and whose third-level components are all elementary formulae or their negations. For example, one simple formula in this form is **(A or B) and (not-C or D)**.

Resolution consists of a single rule, repeatedly applied. The rule simply combines the top-level components of the formula pairwise, selecting components that have an element in common, with the proviso that in one case the element is negated. The rule throws out these conflicting elements and sticks the result together into one component. Thus, **(A or B) and (not-A or C)** resolves to **(B or C)** by throwing out the **A** and the **not-A**. If, after repeatedly banging components against each other this way, we come up empty-handed (components annihilate each other utterly), then the formula is unsatisfiable.

Unfortunately, not all the components are so simple in form; does **P(a)** cancel out **not-P(x)**? Answering that question in the particular case involves a process not unlike finding the lowest common denominator for two fractions and is accomplished by an algorithm called unification. Unification finds a substitution for **x** and **a** that is just sufficiently general to stand for both.

Resolution as originally defined is not an efficient algorithm. In 1965, J.A. Robinson defined an improved resolution algorithm that got around some of the problems of combinatorial explosion that plagued resolution, but this still left unsolved the problem of how to decide which pairs of components to resolve. Current work in AI is emphasizing heuristic strategies that take advantage of knowledge about the domain of discourse in guiding resolution.

(called "pays_too_much_rent") that holds between various data items. The translation of a rule into specific machine actions is the job of the interpreter, not of the programmer.

Prolog Can Be Used to Implement a Meta-language

Prolog is an inherently extensible language because it is possible to modify the behavior of the interpreter within the context of the language itself. Kowalski refers to this phenomenon as the meta-language/object language relationship.

A meta-language controls the interpretation of an object language. A meta-language can be implemented in the object language (Prolog itself) as a program running on top of the interpreter. The meta-language program receives queries and assertions from the user, transforms them in consultation with the knowledge base, then constructs queries in the object language and executes them.

When first confronting Prolog code, many people mistakenly assume that an end user will have to learn how to write Prolog queries in order to use a Prolog program. One way that a meta-language can be used is to give Prolog an alternative syntax; as a result, the user interface to a program is completely specifiable. Another application of a meta-language involves saving the inference path used by Prolog to answer a query, so that the user can see how an answer was generated. This ability to explain how an answer was arrived at is an essential component of expert systems. Note that this is something more than mere extensibility, which allows the programmer to add to the repertoire of the language. Here we are considering the ability of a piece of software to reason about its own reasoning processes.

Applications of Logic Programming

Prolog has been used for artificial intelligence applications as well as more conventional applications, some of which are enumerated below.

Relational Data Bases

Because the relational data base model is a logical, well-developed formalism, it can be represented in Horn clause

form in a straightforward manner. Prolog has also proved particularly useful for data base front ends. Many types of query languages have been implemented in Prolog, including QBE, SQL, relational algebra, and one similar to VisiCalc.

Software Engineering

Kowalski has shown that a program specification can be directly transformed into a logic program, provided that it is written in a logical specification language (such as DeMarco's data flow diagrams). The initial translation produces a top down design. Each piece of the design can be successively refined independently of the other pieces.

Natural Language

Colmerauer originally invented Prolog to perform natural language processing. Warren and Pereira have continued work in this field with the Definite Clause Grammar formalism (a form of top down parser for natural language). More recently, researchers at ICOT have implemented BUP (a bottom up parser for natural language).

Knowledge Representation

Knowledge representation paradigms used in artificial intelligence, such as semantic networks, frames, production rules, and objects, can all be expressed in logic and implemented in logic programming. According to Hayes, semantic networks and frames are essentially alternative syntaxes for logic.

Expert Systems

Once knowledge has been represented, an expert system can put it to use for expert consultation. Expert systems have been built with Prolog in a variety of areas, including equation solving, medicine, law, architecture, factory automation, etc.

Future applications are left as an exercise for the reader.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 192.

GROWING OLD?

waiting
for C programs to
compile and link..?



Use **C-terp**
the complete C interpreter

This is the product you've been waiting (and waiting) for!

Increase your productivity and avoid agonizing waits. Get instant feedback of your C programs for debugging and rapid prototyping. Then use your compiler for what it does best...compiling efficient code ...slowly.

C-terp Features

- Full K&R C (no compromises)
- Complete built-in screen editor--no half-way house, this editor has everything you need such as multi-files, inter-file move and copy, global searching, auto-indent, tab control, and much more.
- Fast--Linking and semi-compilation are breath-takingly fast. (From edit to run completion in a fraction of a second for small programs.)
- Convenient--Compiling and running are only a key-stroke or two away. Errors direct you back to the editor with the cursor set to the trouble spot.
- Compiler Compatible--You can access functions and externals compiled with C86 or Lattice C or assembly language. Utilize your existing libraries unchanged!
- Complete Multiple Module Support--Instant global searches, auto-compile everything that's changed, etc.
- Many more features including batch mode and symbolic debugging.
- Runs on IBM PC, DOS 2.x, 192K
- Price: \$300.00 (Demo \$45.00) MC, VISA

Price of demo includes documentation and shipping within U.S. PA residents add 6% sales tax. Specify C86 or Lattice version.

GIMPEL SOFTWARE

3207 Hogarth Lane • Collegeville, PA 19426
(215) 584-4261

*Trademarks: C86 (Computer Innovations), Lattice (Lattice Inc.), IBM (IBM Corp.), C-terp (Gimpel Software)

Circle no. 38 on reader service card.

INTRODUCING Interface Technologies' Modula-2 Software Development System

The computer press is hailing Modula-2 as "the next standard in programming languages." Modula-2 combines the strengths of Pascal with the features that made C so popular, like independent compilation and direct hardware control.

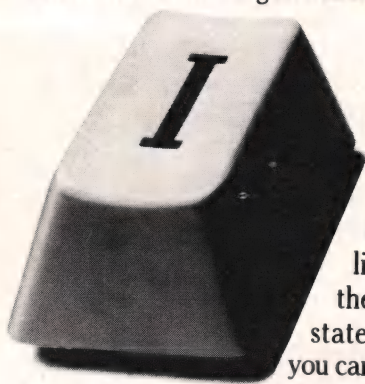
But until today, no company offered a Modula-2 system that made the development of software fast, easy and efficient. Now, though, there's a new tool at your disposal.

The fast, powerful tool for programmers

The breakthrough is here: Interface Technologies' new Modula-2 Software Development System for the IBM® PC, XT, AT and compatible computers to give programmers the same quantum leap in productivity spreadsheets and word processors gave to end-users. It can reduce monotonous wait time, will dramatically increase speed, help stop thoughtless mistakes, and free you to become more creative in virtually all of your programming efforts.

How to speed input and eliminate 30% of errors

Thirty percent of programming mistakes are syntax errors and simple typos in the program structure. Our "syntax-directed" Modula-2 editor does away with these time-consuming headaches once and for all.



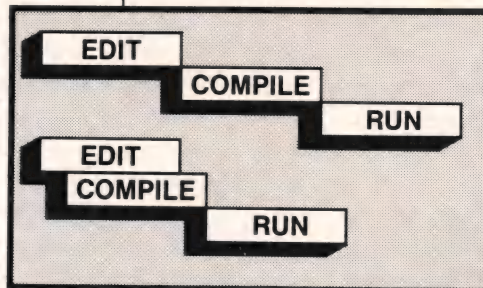
Enter complete statements
with one keystroke.

It speeds input by cutting manual typing as much as 90%, letting you enter statements with a single keystroke. For example, if you type a capital "I" to begin a line, the editor completes the logical "IF THEN" statement automatically, so you can concentrate on what you want to program, rather than concentrate on what you're typing.

The editor locks out errors, finishing statements and procedures in perfect accord with the standardized rules of Modula-2. It also indents and formats your text automatically, making programs easy to read and maintain, an important feature on big projects.

And if you leave an undefined variable or data type, the editor detects the mistake and gives you the option of on-line "help" to correct it. No other programming text editor offers you so much innovation at any price.

How to turn "wait time" into "work time"



It not only has a faster compiler, it also
saves time by compiling while you edit.

The vast majority of programming time is spent waiting, and the biggest slowdown is most often with compilers.

THE ANATOMY OF A

Our compiler turns wait time to work time with a new innovation that lets you compile in the "background."

With background compilation, your program is automatically compiled into object code line by line as you work, every minute you spend writing or editing a Modula-2 program!

When you're finished editing, all that's left for the compiler is a quick mopping up job that generates optimized native code in a single pass.

How quick is "quick"?

Thanks to background compilation and the fact that the compiler itself is so fast, Interface Technologies' compiler turns 100 lines of typical Modula-2 text into optimized machine code in *under five seconds*.

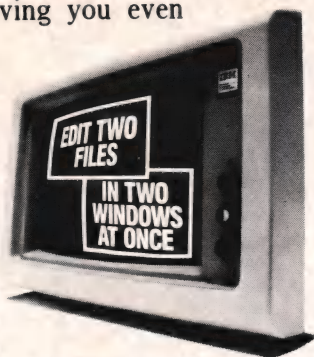
Plus the Interface compiler produces compact code with execution speed superior to that produced by any other Modula-2 compiler on the market.

How to do two things at once

Along with the background compiler and syntax-directed editor, which can save you hours every day and make you more productive, Interface Technologies' Software Development System gives your monitor

windows so you can refer to one file while you edit another simultaneously, saving you even more time.

Concurrent editing of two or more files is especially useful when doing programming work that's intended for separate compilation, and Interface Technologies has the only Modula-2 system on the market that provides you with this helpful benefit for developing software.



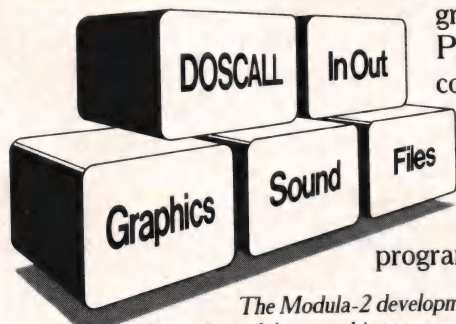
Work with multiple files faster, easier in windows.

How preprogrammed modules speed development

One of the advantages of Modula-2 is that it lets you build large, reliable programs quickly, by linking together many smaller "building-block" modules.

The development system's toolkit of precompiled program modules includes the standard Modula-2 library, and adds exclusive link-and-run modules for direct calls to the operating system, sound, and color

graphics support. Plus you get low-cost updates from the Interface Technologies fast-growing library of new programming modules.



The Modula-2 development system's toolkit of ready-made modules turns big programs into smaller projects.

Increase productivity for \$249

Interface Technologies' Software Development System is fast, powerful and unlimited. It works so well that it's the same tool Interface Technologies is using to write business and consumer applications in Modula-2.

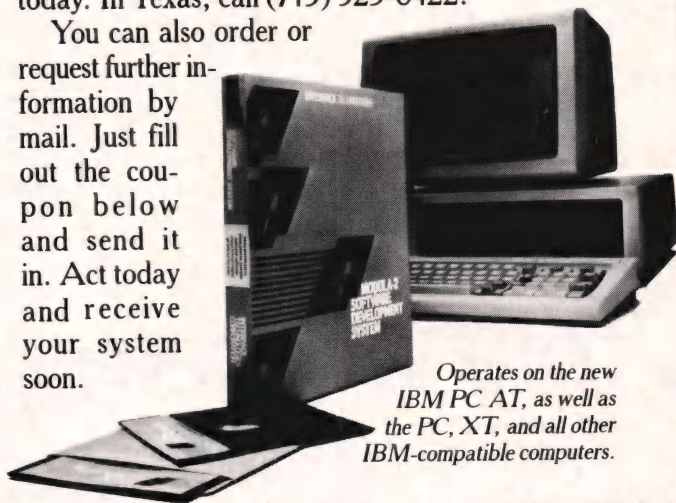
For \$249, you get the syntax-directed editor and compiler, linker, module library and tutorial that will have even modestly experienced programmers writing in Modula-2 in days. And you have full rights to your work; there's no license fee for programs you develop with the Interface Technologies system.

You can use it on any IBM® PC, XT, AT or compatible computer with two double-sided, double-density floppy drives and 320K RAM diskette.

You get a thoroughly indexed, comprehensive user's manual and free telephone support from Interface Technologies. But the most important thing you get is the future, and *the programming language of the future is Modula-2*, and now it's easier than ever.

For more information, or to order the Modula-2 Software Development System, call 1-800-922-9049 today. In Texas, call (713) 523-8422.

You can also order or request further information by mail. Just fill out the coupon below and send it in. Act today and receive your system soon.



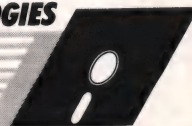
Operates on the new IBM PC AT, as well as the PC, XT, and all other IBM-compatible computers.

BREAKTHROUGH

NAME _____
 ADDRESS _____
 CITY _____ STATE _____ ZIP _____
 PHONE _____
 PLEASE CHECK ONE:
☐ AMERICAN EXPRESS ☐ VISA ☐ MASTERCARD
☐ CHECK ENCLOSED
 CHARGE ACCOUNT NUMBER

EXPIRATION DATE _____ SIGNATURE _____
 PLEASE SEND ME _____ COPIES @ \$249 EACH.
 INTERFACE TECHNOLOGIES CORPORATION
 3336 RICHMOND, SUITE 200, HOUSTON, TX 77098
 Texas residents, add 6.125% Sales Tax. DD/2

INTERFACE TECHNOLOGIES



**MODULA-2 SOFTWARE
 DEVELOPMENT SYSTEM**

IBM is a registered trademark of International Business Machines Corporation.

A Tour of Prolog

by D. E. Cortesi

*The distinction between code and data
in programming is as fundamental as
that between theory and observation in
science—and just as false.*

Prolog is a member of the general class of nonprocedural programming languages or, as I prefer to call them, the *descriptive* languages. There have been descriptive languages before—notably the simulation languages like GPSS—but most are designed to serve specific application domains, while Prolog is intended for general programming. Descriptive languages require an approach to problem solving that is radically different from the procedural approach used with an ordinary language like Pascal or BASIC.

Not many readers know Prolog or have Prolog interpreters to experiment with. When planning this issue of *DDJ*, the editors realized that it would contain several large programs written in Prolog. Because Prolog is so different, they wondered whether the readers would be able to follow the program listings.

I had been exploring Prolog in my spare time, so I volunteered to describe it from a programmer's perspective. This article is not meant to be a complete tutorial (see Clocksin and Mellish¹ for that). It is rather an orientation tour to give you enough exposure to Prolog's style so that you can make sense of the listings in the other articles.

A Caution

Be aware that the various implementors of Prolog don't agree on its syntax and writers on Prolog don't agree on the names of its parts of speech. In this overview, I've cast the examples in a presentation syntax that uses the tricks of professional typesetting for clarity. A sidebar that accompanies the article ("Prolog Syntax," page 46) explores simple mappings to the syntax schemes of common implementations. I've noted where my nomenclature varies from other writers' usage.

Prescribing and Describing

We use the familiar programming languages to *prescribe* the solutions to a set of problems. We build the prescription from two radically different materials: active code and passive data. The passage of time is an implicit part of a prescription. "To solve one of these problems," we tell the computer, "do this, then do that"; *that* can be correctly understood only in the context created by *this*.

All that changes in Prolog. In Prolog, we *describe* the solutions to a set of problems. "A solution to one of these problems," we tell the computer, "looks like this or this or this"; each *this* must be (in principle) a true description of a solution, independent of its place in the program. There is no distinction between code and data. We simply describe what is true about the universe of discourse, using identical mechanisms to describe true facts and true interpretations of the facts.

Assertions and Queries

Let's tie these abstract notions down with an informal example. The problem is one that arises in a card-playing program: defining what a card is. The honor cards are defined as follows:

David E. Cortesi, 430 Sherman Ave. #212, Palo Alto, CA 94306.


```
card(a).
card(k).
card(q).
card(j).
```

These are sentences of fact that we may *assert* (add to the program) at once. We'll deal with matters of syntax later; for the moment you should find it believable that, given those sentences, a Prolog system could conduct a dialogue along these lines:

```
card(k)?
yes.
card(w)?
not so far as I know.
```

This is a conversation with a hypothetical Prolog system. I don't know of any real systems whose dialogue is so natural nor whose negative answer is so humble. Nevertheless, we have here the essence of Prolog operation: a number of sentences are asserted (usually they are loaded from a file); the system is given a pattern to test against those sentences; and it replies "yes" or "no" depending on whether or not it can match the pattern.

It probably occurs to you that it wouldn't take a particularly clever system to hold the example dialogue. A simple comparison loop would do it. Of course, smarter dialogues are possible. For example, all systems allow the query pattern to contain variables as a way of requesting more information. (In this article, I shall indicate variables with italicized words; see the sidebar "Prolog Syntax" for the variety of ways to recognize variables in real systems.) Thus we may ask:

```
card(what)?
yes, what = a.
yes, what = k.
yes, what = q.
yes, what = j.
```

The system finds not just one but four matches to the query `card(what)?` and reports the resulting value of the variable *what* in each case.

Compound Sentences

Let's continue the example by describing the other nine playing cards. Simply to assert nine more facts, `card(10)` through `card(2)`, would be in the spirit of Prolog, but Prolog would be a poor, flat notation if it didn't let us construct sentences in terms of other sentences. We can describe the remaining cards by asserting a single sentence:

```
card(x) if
  integer(x) and
  less(1 x) and
  less(x 11).
```

Although awkwardly phrased, this sentence translates as the rule: "Something *x* is a card if it is an integer between 2 and 10 inclusive." Now we should be able to conduct the following dialogue:

```
card(6)?
yes.
```

This reveals a greater cleverness in the underlying system. To answer the question, the system had to identify 6 with the *x* in our latest sentence, then test the conjunction of three subordinate assertions, verifying the truth of `integer(6)`, `less(1 6)`, and `less(6 11)`. Shortly, we will examine how this is done, but first let's round out the idea of descriptive programming.

Descriptive Programming

The five sentences asserted so far constitute a purely descriptive solution to the problem of defining a card. We say that they define the *card relation*. The sentences, which are independent of each other, together enumerate all possible cases of the card relation. We don't care in which order a Prolog system checks our question `card(6)?` It may test the question against the five sentences in any order; it will turn up the right answer regardless. This has important implications for performance (see the sidebar "Prolog in Parallel" on page 50).

Such pure, timeless description is an ideal rarely achieved in practice; working Prolog programs usually have some dependencies on their sequence (we'll see why later). Nevertheless, pure description should be the goal of a Prolog programmer, just as clean structure and avoidance of side effects are goals when using a prescriptive language.

Descriptive Arithmetic

The implementation of arithmetic in the Micro-PROLOG product² is an example of just how far you can go in the direction of pure description. In that system, all arithmetic is implemented through two relations built into the system. The SUM relation is defined as:

```
SUM(a b s)
```

for all sets of three numbers such that $a + b = s$. In our example, a set of sentences defined the card relation. The SUM relation is implemented so that you could almost believe it was defined by a huge data base of sentences such as:

```
SUM(0 0 0).
SUM(0 1 1).
...
SUM(15 9 24).
...
SUM(6 32761 32767).
```

and so on for all negative and fractional numbers as well. This permits you to use SUM for either addition

```
SUM(6 8 x)?
yes, x = 14.
```

or subtraction

```
SUM(x 8 14)?
yes, x = 6.
```

with equal facility.

Micro-PROLOG's built-in relation PROD is defined as follows:

`PROD(a b c d)`

over all numbers such that $(a * b) = (c + d)$. PROD also is implemented as if the system contained an infinite data base of sentences about numbers: `PROD(16 3 47 1)` and so forth. The PROD relation answers queries needing multiplication, division, or modulus:

`PROD(16 4 prod 0)?`

yes, *prod* = 64.

`PROD(16 quotient 64 0)?`

yes, *quotient* = 4.

`PROD(16 q 65 r)?`

yes, *q* = 4, *r* = 1.

[Extra credit puzzle: If PROD were limited to the domain of integers between 0 and 32767 inclusive—which Micro-PROLOG is not—how many sentences would its imaginary data base contain?]

Let's use SUM to define a relation we'll need later, the predecessor/successor relation:

`succ(x y) if`

`SUM(1 x y).`

We may read this as: "The successor of a number *x* is *y*, the number 1 greater than *x*." The interesting thing is that, because of the descriptive nature of SUM, we can work this relation in either direction:

`succ(7 what)?`

yes, *what* = 8.

`succ(what 14)?`

yes, *what* = 13.

Therefore, the rule may also be: "The predecessor of a number *y* is *x*, the number 1 less than *y*."

Other Prolog systems handle arithmetic relationships much more awkwardly: they attempt to graft the arithmetic expressions of procedural languages onto Prolog syntax. In these systems, the predecessor/successor relation would be defined with syntax such as:

`succ(x y) if`

`y = x + 1.`

Not only does this introduce the whole truckload of reserved operator symbols and operator precedence rules into Prolog—in sharp contrast to the economy of SUM and PROD—but we are not sure whether we can use a sentence of this form to query for a predecessor as well: it is unclear whether the assertion $y = x + 1$ has meaning when worked "backwards." Such a system would require an explicit predecessor rule:

`pred(y x) if`

`x = y - 1.`

Prolog Syntax

Many Prolog interpreters have been built, most of them academic experiments, each with its own features and its own syntax as dictated by the interests and tastes of its developers and the direction of their research. One of the most influential Prolog interpreters was built for the DEC-10 at the University of Edinburgh.³ The best Prolog tutorial text¹ is based on this "Edinburgh syntax." Because of this book, the newer products for personal computers tend to follow the Edinburgh syntax as well. However, it would be unrealistic to expect that any interpreter would follow the syntax exactly; after all, an implementor must display creativity somehow.

Another influential implementation is the Micro-PROLOG product; because it is available for both Z80-based CP/M systems and for MSDOS, Micro-PROLOG has the potential to outnumber all other Prologs put together.

There are only eight syntactic concepts in the Prolog notation: four kinds of items (numbers, variables, constants, and lists) and four higher groupings (clauses, conjunctions, facts, and rules). Even after adding whatever protocols are needed to signal the interpreter that a particular input represents a query or an assertion, you still have a very simple language. Let's see how these eight elements are formed in the two main syntactic traditions.

Although systems should differ little in the formation of literal numbers, they will differ in the magnitude and precision of the numbers supported. Some systems will support only signed integers while others allow reals.

On the other hand, systems will differ dramatically in the rules for forming variables. Edinburgh-style systems take as a variable any token that begins with an initial cap: John is a variable, but john is a constant. This allows a good range of meaningful variable-names, but it is easy to make a mistake and type an initial cap in what should be a constant, as in the assertion

`likes(John,Mary).`

which asserts that anyone likes everybody.

We may class the underscore as a capital letter, so that `_john` becomes a variable. The underscore alone is sometimes an "anonymous variable," a nameless place-holder used to name any argument that is required but not used elsewhere in the sentence.

Micro-PROLOG limits variables to the initial letters "XxYyZz" followed by zero or more digits. This restriction pinches: it makes it difficult to read sentences of more than trivial length. Contrast these versions of the QuickSort sentence:

`qsort([A B C | Tail] Out) if`

`part([A B C | Tail] Left Right) and`

`qsort(Left Sleft) and`

`qsort(Right Sright) and`

`cat(Sleft Sright Out).`

`qsort([X Y Z | x] y) if`

`part([X Y Z | x] z X1) and`


```
qsort(z Y1) and
qsort(X1 Z1) and
cat(Y1 Z1 y).
```

The readability of variables is less important than it might seem at first; apparently no Prolog system retains variable-names after entering a sentence to the data base! When the sentences are listed afterward, they usually have system-generated numeric place-holders in the variable positions. Micro-PROLOG simply restricts input variables to what the system will generate anyway.

After the Prolog system has set the rules for variables and has preempted a set of special punctuation characters, whatever is left is a constant. Uncapitalized words are constants in all systems, but particular systems may use many other tokens as constants as well. Hyphenated words are usually valid constants, although some systems may take mother-in-law as an arithmetic expression. In Micro-PROLOG, certain characters are both constants and delimiters; that is, they cannot be grouped with other characters in larger tokens, but they may be used as constant tokens themselves. The square brackets fall in this category: [alone is a constant, but [a] is interpreted as three constants in a row.

Most systems allow some kind of escape so that a constant can be made from an arbitrary sequence. Edinburgh systems use the single quote ('The Tree' is a constant), while in Micro-PROLOG, the double quote ("The Tree") serves that function. This creates a problem, because in Edinburgh systems the double quote supports a glitch: it indicates a *list* composed of the quoted ASCII characters. Micro-PROLOG achieves the same result with the built-in relation STRING-OF(*c list*), meaning *list* is a list composed of the letters of constant *c*. The Micro-PROLOG relation can be worked backwards, from a list to a constant, but not so the Edinburgh syntactic glitch.

In most Prolog dialects, a list pattern is [*car | cdr*], but in Micro-PROLOG it's (*car | cdr*)—using parentheses. The Edinburgh-influenced systems use the brackets and parentheses to make a distinction:

```
cat([ ] x x).
```

saying, in effect, that a list *as* an argument is different from a list *of* arguments. The Micro-PROLOG syntax doesn't make that distinction (for good reason, as we'll see shortly), so its equivalent formulation would be:

```
(cat ( ) X X)
```

We can convert an Edinburgh-style sentence to Micro-PROLOG form by making a series of simple transformations, one of which is to convert all square brackets to parentheses.

This brings us to the clause, the fundamental way of naming a relationship between items. In this article, I've followed the Edinburgh style, making a clause look like a procedure call in a prescriptive language. That puts the relation-name on a higher level than its arguments, so that

```
length([ ] 0).
```

With a descriptive definition of arithmetic, however, the predecessor relation is simply:

```
pred(y x) if
succ(x y).
```

Unfortunately, the designers of Micro-PROLOG stopped short of implementing all the implications of descriptive arithmetic. Their interpreter insists that the SUM and PROD relations be given enough literal data so that the query has a determinate answer. This means that the question SUM(*x y* 18)? or "what numbers *x* and *y* add up to 18?" won't be answered; the answer is indeterminate because an infinity of number pairs add up to 18. I think these relations should generate random integers for the unknowns, in arbitrary sequence, until they have enough knowns to solve the query. How charming it would be to ask the system PROD(*a b p r*)? and be told "yes, *a* = 2, *b* = 138, *p* = 11, *r* = 265," or some other arbitrary but consistent answer!

Micro-PROLOG's descriptive approach to arithmetic is not important in itself; let it serve as an example of the power and economy made possible with a descriptive style of programming—and as an example of how strange and unsettling such a style initially can seem.

Getting Formal

It's time for some formal nomenclature. Writers on Prolog don't agree on the names of Prolog's parts of speech (any more than its implementors agree on a syntax), so some of these names are my own invention. The named concepts, however, are always present and easy to identify.

A Prolog program consists of a collection, usually called a *data base*, of *sentences* that define *relations*. A sentence's initial word identifies the relation of which it is a part; all the sentences that begin with that word define the relation. In a preceding example, we defined the card relation in five sentences, each beginning with the word "card."

In addition to the relations defined by the programmer, certain fundamental relations are defined by code built into the Prolog interpreter.

A *clause* is a word followed by a parenthesized list of (possibly zero) *items*: card(6), for example. We'll explore the variety of items shortly. (What I call a clause different writers have dubbed a predicate, a function, and an atom.)

Sentences have two forms. A *fact* consists of a single clause:

```
card(k).
rank(k 13).
```

The sense of a fact is that the named relation is always true of the given items. The second form of a sentence is the *rule*. It consists of a clause, an **if** symbol, and one or more clauses separated by **and** symbols:

```
succ(x y) if
SUM(x 1 y).
card(x) if
integer(x) and
less(1 x) and
```


less(x 11).

A sequence of one or more **and**-delimited clauses is called a *conjunction*. The sense of a rule is that the relation is true of these items as long as all of the clauses of the conjunction are also true.

The function of the Prolog interpreter is to answer *queries*. The form of a query is simply a conjunction, usually delimited by some kind of query symbol to trigger the interpreter into action. The interpreter answers a query by verifying each clause within it against the data base. Now, take this slowly: a clause is verified (1) when it matches a fact, or (2) when it matches the clause at the head of a rule *and the conjunction from that rule has been verified as a query*.

In other words, to answer the query card(6)? the interpreter will match the query's first and only clause to the clause at the head of the following rule:

```
card(x) if
  integer(x) and
  less(1 x) and
  less(x 11).
```

As a result, it will pose the conjunction

```
integer(6) and
less(1 6) and
less(6 11)?
```

to itself as a new query.

Zowie! Recursion! This nesting of queries continues until all clauses match facts or system-defined relations or until a clause that can't be matched causes the query to fail. That's the entirety of Prolog execution: the recursive matching of clauses into the data base until all clauses have been verified or until some clause can't be matched.

In principle, there is no defined order for searching the sentences of a relation and no defined order for verifying the clauses of a conjunction. In practice, programmers rely heavily on the expectation that the system will test the sentences of a relation in the order they are asserted, not attempting the second until the first has failed. Also in practice, there is a strong assumption that the clauses of a conjunction will be verified in order from left to right.

Item Types

We've seen almost the whole of Prolog syntax: a relation that is composed of sentences, a sentence that is either a fact or a rule, and a query that is merely a conjunction to be tested one clause at a time. All that remains is to specify what sort of items we may name as the arguments of a clause. There are four: the *number*, the *constant*, the *variable*, and the *list*.

The number needs no explanation; it is simply a literal number. Some Prologs support only integer arithmetic, while others allow floating-point numbers.

The constant is a notion that may be unique to Prolog. A constant is a literal word. It doesn't stand for anything, being neither the name of something nor a character array or string. The value of a constant is simply itself, just as the value of 6 is 6, the value of "a" is "a," and the value of

seems to say "length is true of" In fact, this is a false distinction, albeit a harmless one. The interpreter's matching process does not depend on the form of the clause; it would work just as well if the relation-name were inside the parentheses:

```
(length [ ] 0).
```

Matching is just a comparison of items from left to right.

Clauses are combined to make conjunctions, sequences of clauses that are true only if all their component clauses are true. A query is a conjunction. A rule is a clause plus the conjunction that qualifies it, and a fact is a clause that needs no qualification. There is an implicit *and* between each clause of a conjunction and an implicit *if* between the head of a rule and its conjunction.

Whoever devised the Edinburgh syntax established the colon-hyphen (:-) as the if symbol, set the comma as the and symbol, and required a period at the end of an asserted sentence (but a question mark at the *start* of a query!). Thus, the Edinburgh form of length is

```
length([ ] 0).
length([Head|Tail] Len) :-
  length(Tail Tlen),
  succ(Tlen Len).
```

Some systems, however, may not permit multi-line input.

The Micro-PROLOG syntax for clauses, conjunctions, and rules is rigorously logical but even less friendly:

```
((length ( ) 0))
((length (X|Y) Z)
 (length Y x)
 (succ x Z))
```

In this inner, or fundamental, syntax of Micro-PROLOG, everything is lists. A clause is a list whose first item is its naming constant. A sentence is a list of clauses. We can formalize these elements using the presentation syntax of this article:

```
clause([name|args]) if
  constant(name).
```

```
conjunction([ ]).
conjunction([c|etc]) if
  clause(c) and
  conjunction(etc).
```

```
sentence([head|conj]) if
  clause(head) and
  conjunction(conj).
```

Undoubtedly, Edinburgh-style interpreters store asserted sentences in a similar nested-list arrangement; they simply put a nice front end on it.

So does Micro-PROLOG, except that it uses a variety of front ends, each written in Prolog. Because the system allows

APROTEK 1000™ EPROM PROGRAMMER



only
\$250.00

TECHNICAL
BREAKTHROUGH
NOW ALLOWS A
PRICE
BREAKTHROUGH

A SIMPLE, INEXPENSIVE SOLUTION TO PROGRAMMING EPROMS

The **APROTEK 1000** can program 5 volt, 25XX series through 2564, 27XX series through 27256 and 68XX devices plus any CMOS versions of the above types. Included with each programmer is a personality module of your choice (others are only \$10.00 ea. when purchased with **APROTEK 1000**). Later, you may require future modules at only \$15.00 ea., postage paid. Available personality modules: PM2716, PM2732, PM2732A, PM2764, PM2764A, PM27128, PM27256, PM2532, PM2564, PM68764 (includes 68766). (Please specify modules by these numbers).

APROTEK 1000 comes complete with a menu driven BASIC driver programmer listing which allows READ, WRITE, COPY, and VERIFY with Checksum. Easily adapted for use with IBM, Apple, Kaypro, and other microcomputers with a RS-232 port. Also included is a menu driven CPM assembly language driver listing with Z-80 (DART) and 8080 (8251) I/O port examples. Interface is a simple 3-wire RS-232C with a female DB-25 connector. A handshake character is sent by the programmer after programming each byte. The interface is switch selectable at the following 6 baud rates: 300, 1.2k, 2.4k, 4.8k, 9.6k and 19.2k baud. Data format for programming is "absolute code" (i.e., it will program exactly what it is sent starting at EPROM address 0). Other standard downloading formats are easily converted to absolute (object) code.

The **APROTEK 1000** is truly universal. It comes standard at 117 VAC 50/60 HZ and may be internally jumpered for 220-240 VAC 50/60 AZ. FCC verification (CLASS B) has been obtained for the **APROTEK 1000**.

APROTEK 1000 is covered by a 1 year parts and labor warranty.

FINALLY - A Simple, Inexpensive Solution To Erasing EPROMS

APROTEK-200™ EPROM ERASER

Simply insert one or two EPROMS and switch ON. In about 10 minutes, you switch OFF and are ready to reprogram.

APROTEK-200™ only \$45.00.

APROTEK-300™ only \$60.00.

This eraser is identical to **APROTEK-200™** but has a built-in timer so that the ultraviolet lamp automatically turns off in 10 minutes, eliminating any risk of overexposure damage to your EPROMS.

APROTEK-300™ only \$60.00.

APROPOS TECHNOLOGY

1071-A Avenida Acaso, Camarillo, CA 93010

CALL OUR TOLL FREE ORDER LINES TODAY:

1-(800) 962-5800 USA or 1-(800) 962-3800 CALIFORNIA

TECHNICAL INFORMATION: 1-(805) 482-3604

Add Shipping Per Item: \$3.00 Cont. U.S.

\$6.00 CAN, Mexico, HI, AK, UPS Blue

Circle no. 8 on reader service card.

GREAT PROGRAM . . . TERRIBLE MANUAL . . .

How often have you seen or heard that said about software?

If you have a first-class program you deserve the **BEST** documentation. At **A/N SOFTWARE** we can provide you with a top-quality manual in the shortest possible time.

Whether it's a complete service from writing, to design, artwork, typesetting, mechanicals, and printing, or any step along the way, your manual will be in a class with the best documentation available today.

Our staff consists of writers, artists, testers, and even includes a CPA. Business and technical programs are a specialty. We pride ourselves on designing manuals that look expensive, but aren't.

We understand the time constraints of the software business; we'll meet your deadlines.

We're proud of our work and would like to send you samples. For additional information, samples, or just some advice from the experts call:

. . . 516-549-4090 . . . or write



SOFTWARE Inc.

P.O. Box 895

Melville, NY 11747

Circle no. 5 on reader service card.

WIRED?



WIRELESS FILE TRANSFER.

APPLE TURNOVER™

A "wireless file transfer" package for the IBM PC® to Apple II and back, and back again. Apple turnover is a firmware board which fits into any slot in the IBM PC® or compatible, and software running under MS-DOS®. No modems, no serial links, no hassles, no problems. **APPLE TURNOVER™** will format Apple CP/M® and Apple DOS 3.3 disks. Leave your IBM and Apple computers where they are. Simply bring your Apple disk to work and transfer your file to an MS-DOS disk. Allows for minor modifications to text and data files. It's a simple, inexpensive, fast high performance alternative to complicated serial links and modems.

XENO-COPY PLUS™

A "wireless file transfer" software program for your IBM PC and most PC look-a-likes. **XENO-COPY PLUS™** copies files between IBM-PC® and many CP/M® and foreign MS-DOS® disk formats. An uncomplicated and inexpensive way to transfer text and data files. **XENO-COPY PLUS™** allows you to format, copy from and write to nearly eighty different formats. Also, allows for minor modifications to text and data files. **XENO-COPY PLUS™** can be upgraded to **XENO-DISK™** for the price difference.

XENO-DISK™

The high performance model of **XENO-COPY PLUS™**. **XENO-DISK™** formats, writes to, and copies from over 100 different disk formats including 40 and 80 track 5¼" disks. **XENO-DISK™** supports the use of several 8" disk formats. **XENO-DISK™** contains a powerful table driven text translator, "Text-Tran." For low volume disk production, **XENO-DISK™** includes a track-by-track disk duplicator (which is faster than file by file duplication). Gives you the option to input disk format parameters which allows you to utilize uncommon disk formats. **XENO-DISK™** includes a back-up disk and one free update.

80Mate™

A CP/M® 80 emulator for MS-DOS® computers. After programs have been transferred onto PC/MS DOS® disks with **XENO-COPY PLUS™**, **XENO-DISK™**, or **APPLE TURNOVER™**, **80Mate™** lets you simulate most CP/M® 80 systems on your MS-DOS® computer. Includes all internal CP/M® commands and many available functions. **80Mate™** includes a terminal emulator for 7 predefined terminals including **APPLE CP/M®**. You can also input parameters for other terminals that need to be emulated, without additional Apple hardware.

See your dealer or
Call for information:
(213) 938-0857

Vertex
systems, inc.
Innovation in microcomputer products

6022 W. Pico Blvd., Los Angeles, CA 90035

APPLE TURNOVER, XENO-COPY PLUS, XENO-DISK, 80Mate are registered trademarks of Vertex Systems, Incorporated. • IBM PC & PC-DOS is a registered trademark of International Business Machines Corporation. • APPLE is a registered trademark of Apple Computers Inc. • MS-DOS is a registered trademark of Microsoft Corporation. • CP/M is a registered trademark of Digital Research Incorporated.

Circle no. 78 on reader service card.

"card" is "card."

The function of the constant, of course, is to supply a comfortably large set of atomic symbols that need no further interpretation so we can write interesting facts in human-readable form. That different systems have different rules for forming constants is a major irritation (see the sidebar "Prolog Syntax").

The variable is a syntactic place-holder; it means "whatever matches to this position." In these hypothetical examples, I'm italicizing variables, but we obviously cannot do that with real Prologs. The syntactic space of word-like objects must be divided between constants and variables. Because different implementors draw the line in different ways, a variable in one system may be a constant in another.

List Notation

Most Prologs support the list data structure of Lisp (indeed, most Prologs are essentially special bodies on a Lisp chassis). However, a Prolog system does not construct lists; it *describes* them, and the syntax for describing a list is central to most Prolog applications.

A list is described in full by a list of items in brackets, so that [a k q [8] 10] is a list of five items (three constants, a list, and a number), while [] is the empty list. (Most Prologs use special brackets like these. One of Micro-PROLOG's less endearing features is that it uses ordinary parentheses.)

Often it is convenient to describe only the head of a list, referring to the remainder of the list with a variable. This is done with a *list pattern* that employs the *list constructor symbol*, usually the stile (|). Thus, the pattern [6|etc] is a description of all lists having the number 6 as their first item. The pattern describes the first item exactly but says nothing about the remainder of the list except that the variable *etc* stands for it.

Now, mark two crucial rules. First, *the tail of a list is a list*. Thus, when the pattern [6|etc] is matched to the list [6 5 4], *etc* stands for [5 4], a list. Should it be matched to [6 5], *etc* would stand for [5]—the list [5], not the number 5.

Second, *the empty list is the tail of all lists*. This rule supplies a tail for lists that have but one visible element, enabling the matching of list patterns to be more consistent. Thus, the pattern [6|etc] describes the one-item list [6], with *etc* standing for [], the empty list that is the tail of all lists, even short ones.

From these rules come certain common effects. First, the empty-list pattern, [], matches only the empty list; no other pattern does this. It is really the only special case among lists.

Second, the pattern [head|tail] matches any nonempty list. A list pattern with a single variable preceding the list constructor matches any list that contains at least one item.

In fact, the pattern [a k next|etc] matches any list that contains at least three items, the first two being the constants a and k and the third being unspecified. The variable *etc* is identified with whatever items follow those three items (or with the empty list if there are no more).

Recursive Descriptions

How many items does a list contain? Let's describe the solutions to that problem in an example of real Prolog programming. We could enumerate all possible solutions:

the meta-variable, Prolog programs can construct and then execute other Prolog programs. In essence, Micro-PROLOG's syntax is protean: it can be whatever you wish to disguise it as. The friendliest of the distributed front ends is called SIMPLE; its form of clauses and rules is much like the presentation syntax used in this article.

Prolog in Parallel

There is an implicit *or* between the sentences that define a relation. Take the following card relation:

```
card(a).
card(k).
card(q).
card(j).
card(x) if
    integer(x) and
    less(1, x) and
    less(x, 11).
```

Together these sentences say that a card is this *or* this *or* ... this. The Prolog implementation must search through the statements to answer a question about a card, but (in principle) the order in which it tests them doesn't matter. If hardware is available to make concurrent tests of all five sentences in parallel, fine. Such parallelism is called Or-parallelism because of the implicit *or* between sentences.

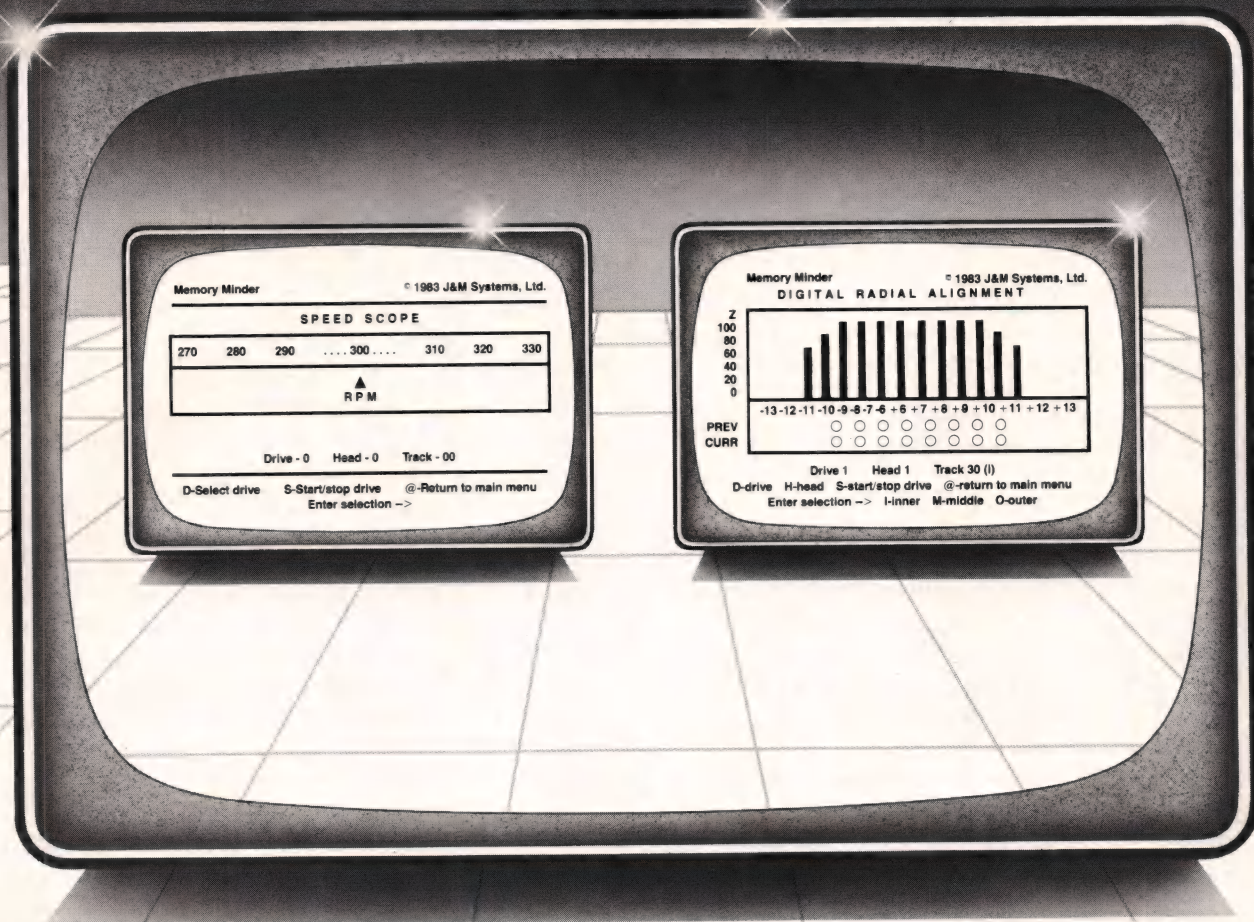
Notice, too, in the last sentence, that there is (in principle) no preferred order for testing the three subordinate assertions. Proposed Prolog systems would evaluate all three assertions concurrently. This parallelism is called And-parallelism because of the *and* relation between the clauses of a conjunction. Either kind of parallelism (properly called concurrency) or both could be used to speed Prolog execution. The Japanese "Fifth Generation" project proposes to explore such highly parallel architectures for Prolog-like languages.

Notice, however, the careful use of "in principle" in the preceding paragraphs. We can express many algorithms, like the card relation and most of the other examples in this article, as sets of pure, achronic, descriptive sentences. Such descriptions executed on concurrent hardware would produce the same results as those produced under a single-thread interpreter. But it is not always easy to see the time-free, descriptive solution to a problem (especially when your mind is thoroughly steeped in the if-then patterns of procedural notations); sometimes there is no solution.

For instance, no nondeterministic description exists of a program that prompts its user and then takes input from the keyboard. There is some time t1 when no prompt is visible, some time t2 when a prompt is visible but no response has been typed, and some time t3 when the response is complete—these three times are *not* the same. Such a program *must* recognize the flow of time. Otherwise, it theoretically could read nonexistent input and prompt afterward.

As noted in the main article, similar time dependencies arise when programs assert and retract facts dynamically. To make such programs work, the available Prolog interpreters promise to evaluate the clauses of a conjunction in

Memory Minder Detects Long Term Disk Drift ... to protect your data integrity



MEMORY MINDER® from **J & M Systems** is the most comprehensive disk diagnostic program available for microcomputers. It allows you to run a testing program on your disks on a regular basis to assure correct specifications.

All disk drives can drift away from their factory specs. You may not be aware of this until you try to read vital data from an older disk, or try to utilize your data disk in another computer. Worse, once head alignment is corrected, you may not be able to read improperly recorded disks!

Preventative Maintenance for Vital Data

With **MEMORY MINDER** you can easily check long-term head alignment drift, index hole timing, spindle speed, hysteresis, azimuth, eccentricity, and other parameters. You can correct these problems before they cause difficulties.

Interchange Error Prevention

Often interchange errors are the result of improperly aligned heads, erratic or improper spindle

speed, or azimuth error. This is a hardware problem which is not only immediately monitored by **MEMORY MINDER** but is often easily corrected while the program is running!

Don't Gamble With Your Records

Ensure top performance all the time by putting **MEMORY MINDER** through its paces at least once a week! Quick, menu driven, user friendly, yet packing the power to make real-time measurements on your disk drives at your workstation. For perfect alignment and positive data retention, you must have **MEMORY MINDER** working for YOU!

Available for IBM®, most IBM compatibles, TRS-80®, Kaypro®, Sanyo® computers from \$79.

Circle no. 24 on reader service card.

J & M SYSTEMS, LTD.

15100-A Central SE
Albuquerque, NM 87123
505/292-4182


```
length( [ ] 0 ).
length( [ x ] 1 ).
length( [ x y ] 2 ).
```

That, however, gets tedious very quickly. A general approach to problem descriptions of this sort is to enumerate only the limiting case(s) then to describe the general case recursively. If we apply that approach to this problem, we soon arrive at:

```
length( [ ] 0 ).
length( [ h | t ] 1 ) if
    length( t k ) and
    succ( k 1 ).
```

In English this says: "The length of the empty list is zero, while the length of a nonempty list is one more than the length of its tail."

Please examine the two Prolog sentences that define the length relation. Do you understand them? Can you see how they are sufficient to permit the dialogue

```
length( [ a k q ] len )?
yes, len = 3.
```

to take place?

If they seem utterly opaque, that might be for either of two reasons. You may not be familiar with the use of recursion to solve problems. If you are baffled by the idea of describing length in terms of length, turn to the sidebar "The Recursive Method" (at right). It covers a general approach to recursion and may get you over the initial hump.

If you are comfortable with the idea of recursion but can't see how a Prolog implementation might handle the given sentences, turn to the sidebar "Executing a Query" on page 54. Watching the Prolog system work its way to the root of a query may clarify these issues.

List Operations

Let's do one more fundamental list operation, the member relation. This is the relation that is true (verifiable) if a particular item is a member of a given list. We wish to be able to ask such questions as:

```
member( j [ k j 9 3 2 ] )?
yes.
member( a [ k j 9 3 2 ] )?
not so far as I know.
```

The recursive method works well here. Because Prolog makes the head item of a list immediately visible, one limiting case is that in which the desired item is visibly the head of the list:

```
member( item [ item | etc ] ).
```

We may read this as: "Some item *item* is a member of any list that begins with *item*."

[This use of variables troubled me at first. It seemed too easy! I wanted to write something like

order from left to right. Programs that rely on that promise could never be executed on And-parallel hardware. In my opinion, there should be a Prolog equivalent of Ada's and-then operator to make time dependencies within a conjunction explicit.

Other cases—too elaborate to go into here—make it seem essential to test the sentences of a relation in a particular order. Present interpreters number the sentences of a relation in the sequence they are entered and promise to evaluate them in that sequence. Programmers writing code that relies on that promise are operating sometimes from necessity but more often from procedural habit. Such code will not execute reliably on Or-parallel hardware.

The Recursive Method

There are two ways to completely express a lengthy process in short terms: iteration and recursion. Iteration—the for, while, and until loops and other variations on the theme of "go back and do it again"—is a customary part of any procedural language. Thus,

```
for i := 1 to 10 do
    sum := sum + a[i];
```

is a short but complete summary of a 10-step process of summation. We may summarize the iterative design approach as: (1) decide how to solve the problem for one item and (2) decide how to repeat that solution for all the other items.

Prolog supports no iteration. Fortunately, many problems will yield to a recursive approach. We may summarize the recursive design approach this way: (1) enumerate the simple cases for which a solution is immediately known; (2) specify how to reduce a complicated case to one that is slightly simpler; (3) specify a general solution in the terms "if I knew the answer to the next-simpler case, the answer to *this* case would be . . ."

Because step (2) supplies the means to generate the next-simpler case, and because repeated application of step (2) must eventually produce a case for which step (1) has defined an answer, we may write the general solution as "return the result of (some small function) on the result returned by applying this general solution to the next-simpler case."

Raising a number to an integral power is an example of a problem that we can handle this way. We want to describe the solutions to

```
power( x p z )
```

such that *z* is the result of raising *x* to the *p*th power.

Step (1): a simple case for which we know the solution immediately is

```
power( x 0 1 ).
```

or anything raised to the zeroth power is 1. The complicated cases comprise all those with *p* greater than zero. (Actually, we know that $\text{power}(x\ 1\ x)$, but it turns out we don't need the knowledge.)

Step (2): to specify how to move the complicated cases one step in the direction of being a simpler case, we simply need to reduce p by 1, thus moving it one step toward the simple case of $p = 0$.

Step (3): applying the magic incantation “if I knew the answer to the next-simpler case, the answer to *this* case would be . . .,” we can say if we knew the answer to $\text{power}(x \ p - 1 \ x\text{-to-the-}p\text{-less-one})$, we could multiply it by x to yield $\text{power}(x \ p \ x\text{-to-the-}p)$. Or in Prolog:

```
power(x p z) if
  sum(1 simplerp p) and
  power(x simplerp simplerz) and
  prod(x simplerz z 0).
```

Any recursive design should be tested for termination; an infinite regression is not a valid answer. Will the nesting ever cease? The formulation for power is unsafe in this regard. If it is queried with p as an integer greater than zero, it will work; each evaluation will reduce p by one step toward the simple case of $p = 0$, which it must ultimately reach. But query it with a fractional p , and it will skip right past $p = 0$ and continue on toward the horizon—as it will if it is queried for any p less than zero. To play it safe, the clauses

```
integer(p) and
less(0, p) and
```

should be inserted.

The speed of an iterative solution is proportional to the number of iterations. Similarly, the speed of a recursive solution is proportional to the depth of its nesting. The power algorithm just given nests p deep. Another recursive design for power nests to drastically fewer levels; math fans may enjoy following it:

```
power(x 0 1).
power(x 1 x).
power(x p z) if
  less(1 p) and
  integer(p) and
  prod(2 q p r) and
  power(x q xbyq) and
  prod(xbyq xbyq xevn 0) and
  condmul(x xevn r z).
condmul(x xevn 0 xevn).
condmul(x xevn 1 z) if
  prod(x xevn z 0).
```

The main difference is that it uses a faster way to march the general case toward the simple case, dividing it by two rather than reducing it by one. This formulation also illustrates one way to handle a conditional operation in Prolog without relying on statement sequence.

Recursive solutions most often are needed to describe lists of any length. The limiting cases typically involve the empty list or the list of one item. The method of moving the general case toward the simple case is to shorten a list by dropping its head.

```
member(item [x | etc ]) if
  equal(item x ).
```

to make the comparison explicit. But that impulse is a hangover from the habits of procedural languages, where comparison is an “operation” to be “executed” upon “data.” In Prolog, we don’t write a *test for equality*; we write a *description of equality*. The variable-binding rules of Prolog interpreters are so implemented that the more succinct statement works. Consider matching query $\text{member}(\text{fred} [\text{mack} \text{pete}])$? versus the sentence $\text{member}(\text{item} [\text{item} | \text{etc}])$. The matching process begins with the binding of the constant “fred” to the variable *item*. Instantly, the remainder of the question becomes:

```
...[mack pete])?
...[fred | etc ]).
```

This soon will result in detection of a mismatch.]

We now have established the limiting case. In the general case—when the desired item isn’t visible as the head item—we must hope that it appears somewhere in the tail of the list. If it does, it must be the head of some portion of the tail. So we drop the present head and try again:

```
member(item [head | rest ]) if
  member(item rest ).
```

This passes the termination test: each application of the general rule talks about a shorter list. The process sooner or later must either bring about the limiting case or else pose a query, $\text{member}(\text{item} [])$, that will match nothing and cause the whole query to fail.

This formulation of the membership problem will work, but it is not perfect Prolog. The two sentences aren’t independent; certain queries will match both of them. I have followed up the implications of that in the sidebar “Cutting and Failing” on page 58.

Having It Both Ways

Conventional (procedural) programs are time-bound and linear, so they can flow in one direction only. Long use of procedural languages makes this seem a natural and inevitable part of programming, but it is neither. We’ve already noted how, when arithmetic is implemented in a descriptive way, a single relation SUM supplies both addition and subtraction. When we based a successor relation on it, the identical relation served to describe the predecessor relation as well—the predecessor being simply the successor “in reverse.”

This ability to work Prolog relations backward seems strange and upsetting at first. It takes a conscious effort to recall that a reverse use is even possible and to check for it. For example, a programmer who is oriented toward procedural languages inevitably will identify the member relation with a Pascal or C function that searches for one item in a list. Such a function can be used in those languages only for that purpose, but this is not so of the member relation in Prolog. Sure, Prolog will test to see if a member exists in a list, but it is also perfectly happy reporting what members do exist in a list:


```
member(x [fred bob sam])?
yes, x = fred.
yes, x = bob.
yes, x = sam.
```

Furthermore, Prolog can be made to generate an infinite number of lists that contain a given item. The query `member(x what)?` or "What's a list containing something?" leads to an infinite recursion producing successive answers:

```
yes, what = [x|etc].
yes, what = [head x|etc].
yes, what = [head head x|etc].
```

Not a useful result, but a strange and beautiful one.

Showing Off

We could use any number of enjoyable list problems for further practice: reversing them, extracting the first or last n items, catenating them (an especially interesting Prolog algorithm that can be worked backwards in several useful ways). But space is limited, so let's glance instead first at programs for sets and trees and finally at a sort.

A set is a list that contains no duplicates. Let's first define a relation `undup(in out)` such that *out* is the list *in* with its duplicate items removed:

```
undup([ ] [ ]).
```

i.e. the empty list has no duplicates;

```
undup([h|t] [h|tnew]) if
  remove(h t tless) and
  undup(tless tnew).
```

i.e. other lists are undup'd by removing all instances of the head from the tail then undup-ing the tail.

Of course, we now require a definition of the relation `remove(x in out)` such that *out* is the list *in* with all occurrences of item *x* removed:

```
remove(x [ ] [ ]).
```

i.e. removing *x* from the empty list yields the empty list;

```
remove(x [x|t] tnew) if
  remove(x t tnew).
```

i.e. removing *x* from a list headed by *x* produces the tail of that list with *x*'s removed;

```
remove(x [h|t] [h|tnew]) if
  not-same(x h) and
  remove(x t tnew).
```

i.e. removing *x* from a list headed by something else (*h*) produces a list still headed by *h* but with *x*'s removed from its tail.

Naturally, it's easiest to build a set from scratch, simply banning all duplicate additions to it. Let's define the relation `addset(x in out)` such that *out* is the set *in* with item *x* added:

For example, the normal Prolog list-pattern syntax makes it easy to describe the first element of a list (*head* in the pattern `[head|tail]`), but there is no primitive way to describe the last item in a list. Can we describe a relation `last(list item)`, meaning that *item* is the last thing in *list*?

Let's apply the recursive method. The empty list isn't a factor here because it has no last (or any other) item. The shortest list having a last element is the one-item list; we may at once assert:

```
last([only] only).
```

This means "the last item in the list `[only]` is the item *only*."

A way of simplifying a list is to drop its head. If we knew the last item in the tail of a list, we would know the last item in the whole list:

```
last([x y|etc] item) if
  last([y|etc] item).
```

The solution must pass the termination test: does it advance toward the limiting case? Yes, each application of that sentence poses a new query about a list that is shorter by one item. Ultimately, it will pose a query about a one-item list, for which we have a fact.

A good Prolog solution should pass two more tests: independence and completeness. Are the two sentences independent; that is, do they describe distinct situations? These do, but only because the second one explicitly describes lists of two or more items. If we had written

```
last([head|tail] item) if
  last(tail item).
```

then a one-item list could match to either rule. Are the sentences complete? Yes, the first covers one-item lists and the second covers all those with two or more items; thus, they cover all lists that have a last item. These two sentences constitute a good, recursive Prolog description of the last-item-of-a-list problem.

Executing a Query

A Prolog query is a conjunction or a list of clauses each of which is to be verified. Let's observe the actions of the interpreter as it executes the query

```
(1) length([a q j] what)?
```

against a data base of this length relation:

```
length([ ] 0).
length([head|tail] len) if
  length(tail tlen) and
  sum(tlen 1 len).
```

A query is verified by verifying its clauses; a clause is verified by matching it to the facts and rules of the data base.

THE PROGRAMMER'S SHOP™

helps compare, evaluate, find products. Straight answers for serious programmers.

SERVICES

- Programmer's Referral List
- Dealer's Inquire
- Compare Products
- Newsletter
- Help find a Publisher
- Rush Order
- Evaluation Literature free
- Over 500 products
- BULLETIN BOARD - 7 PM to 7 AM 617-826-4086

"Guaranteeing" Software - Possible?

It is a very messy issue. Trust is required; but trust means that the publisher, retailer or user gets left open to losing, being hurt. The Programmer's Shop trusts customers and **guarantees SOLID VALUE.** Ask about it. The next "Programmer's Letter" offers ideas.

RECENT DISCOVERIES

ProYAM Communications Package - everything a programmer would want. TTY, VT 100, 3101, MODEM 7, BBS. Remote, macros, windows. MSDOS \$139.

ARTIFICIAL INTELLIGENCE

EXSYS - Expert System building tool. Full RAM, Probability, Why, Intriguing, serious. PC DOS \$295

GC LISP - "COMMON LISP", Help, tutorial, co-routines, compiled functions, thorough. PC DOS \$475

IQ LISP - MACLISP & INTERLISP. Full RAM. Liked. PC DOS \$155.

TLC LISP - "LISP-machine"-like, all RAM, classes, turtle graphics 8087 for CP/M-86, MSDOS \$235

TLC LOGO - fast, classes. CPM \$139

PROLOG-86 - Learn fast, Standard, tutorials, samples of Natural Language, Exp. Sys. MSDOS \$125

Expert System front-ends for PROLOG: APES (\$275), ES/P (\$1895)

Other solid alternatives include: MuLISP-86 (\$250), WALTZ LISP for CPM (\$159), MicroPROLOG (\$275).

EDITORS FOR PROGRAMMING

BRIEF Programmer's Editor - undo, windows, reconfigurable, macro programs, powerful. PC DOS \$195

VEDIT - well liked, macros, buffers, CPM-80-86, MSDOS, PC DOS \$119.

MACINTOSH

We evaluate, carry every available programmers product. Ask.

C LANGUAGE

INSTANT C - Interactive development - Edit, Source Debug, run. Edit to Run - 3 Secs. MSDOS \$500

"INTRODUCING C" - Interactive C to learn fast. 500 page tutorial, examples, graphics. PC DOS \$95

MEGAMAX C - native Macintosh has fast compile, tight code, K&R, toolkit, .OBJ, DisASM MAC \$295

CROSS COMPILERS by Lattice, CI. VAX to 8086. VMS \$3000

C LIBRARIES

COMMUNICATIONS by Greenleaf (\$149) or Software Horizons (\$139) includes Modem7, interrupts, etc. Source. Ask for Greenleaf demo.

C SHARP Realtime Toolkit - well supported, thorough, portable, objects, state sys. Source \$600

PORTABLE C-LIB: Same calls for IBM, Ile, CP/M, C64, more. Screen, I/O, Graphic, more. \$125.

ROMPack - special \$Main .EXE editor, source, tech support, 8086. \$195

DEBUGGERS

PERISCOPE DEBUGGER - load after "bombs", symbolic, "Reset box", 2 Screen, own 16K. \$295

SOURCE PROBE by Atron for Lattice, MS C, Pascal. Windows single step, 2 screen, log file. \$395

FORTRAN LANGUAGE

RM/FORTRAN - Full '77, big arrays, 8087, debugging, xref. MSDOS \$525

DR/Fortran-77 - full ANSI 77, 8087, overlay, full RAM, big arrays, complex NUMS., CPM86, MSDOS. \$249

OTHER LANGUAGES

ASSEMBLER - ask about Microsoft MASM-86 (\$125) improvements or its new competitors.

"BASICA COMPILER": Better BASIC all RAM, modules, structure. \$185

HS/FORTH - '79 & '83 Standards, full RAM, ASM, BIOS, interrupts, graph, multi-task, optimizer MSDOS \$250.

MBP COBOL has screen control, strong doc, '74 interm., fast. MSDOS \$695.

SUPPORT PRODUCTS

BASIC DEVELOPMENT SYSTEM - (BDS) for BASICA; Adds Renum, crossref, compress . . . \$115.

PLINK-86 for Overlays, most lang., segment control. MSDOS \$345

PROFILER-86 - faster programs with less work. Learn quick, symbolic, All Lang. MSDOS \$125

SCIL - Source Librarian to manage Versions, Doc, Minimize disk space, confusion. MSDOS \$335

"C" LANGUAGE

	OUR PRICE
MSDOS: C86-8087, reliable	call
Instant C - Inter., fast, full	500
Lattice 2.1 - improved	call
Microsoft C 2.x	329
Williams, debugger, fast	call
CPM80: Ecosoft C-now solid, full	125
BDS C - solid value	225
MACINTOSH: Softworks	365
Megamax-object, full	295
Consulair's MAC C	295

Compare, evaluate, consider other Cs

BASIC

	RUNS ON
Active Trace-debug	86/80 75
BASCOM-86 - MicroSoft	8086 279
BASIC Dev't System	PC DOS 115
BetterBASIC - 640K	PC DOS 185
CB-86 - DRI	CPM86 439
Prof. BASIC Compiler	PC DOS 89
Databurst - screens	MSDOS 215
SCREEN SCULPTOR	PC DOS 119

Ask about ISAM, other addons for BASIC

SERVICE

ADVICE - We help choose a product to fit your work. Even if you don't know of a product, we'll help.

EDITORS Programming

	RUNS ON	OUR PRICE
BRIEF - Intuitive, flexible	PC DOS	195
C Screen with source	86/80	75
Epsilon - like EMACS	PC DOS	195
FINAL WORD - for manuals	86/80	215
MINCE-like EMACS	PC/86	149
PMATE-powerful	8086	195
VEDIT - full, liked	86/80	119

UNIX PC

COHERENT - for "C" users	PC like	475
COHERENT-NCI-Realtime	PC like	call
XENIX - plus C to MSDOS	PC	1295

Ask about run-times, applications, DOS compatibility, other alternatives. UNIX is a trademark of Bell Labs

LANGUAGE LIBRARIES

GRAPHICS: GraphiC-source in C	MSDOS	195
GRAPHMATIC-3D. FTN. PAS	PC DOS	125
HALO-fast, full-all lang	PC DOS	175
FILE MGMT: Btrieve-all lang	MSDOS	215
Cindex - source, no royal	86/80	375
CTree-source, no royal	ALL	375
dbC ISAM by Lattice	8086	250
PHACT-up under UNIX, addons	MSDOS	465
OTHER: CUtil by Essential	MSDOS	250
Greenleaf - 200 +	MSDOS	139
CSharp - Real-Time	MSDOS	165
PORTABLE C to PC, Mac, II	MSDOS	600
SOFT Horizons - Blocks I	Many	125
SCREEN: CURSES by Lattice	PC DOS	139
Cview - input, validate	PC DOS	125
MetaWINDOW - icons, clip	PC DOS	195
PANEL - many lang, term	PC DOS	139
	MSDOS	265

FORTRAN

	RUNS ON	OUR PRICE
MS FORTRAN-86 - Impr.	MSDOS	\$ 239
DR Fortran-86 - full '77	8086	249
PolyFORTRAN-XREF, Xtract	PC DOS	165

OTHER PRODUCTS

Assembler & Tools - DRI	8086	159
Atron Debugger for Lattice	PC DOS	395
cEnglish - dBase C	MSDOS	795
C Helper: DIFF, xref, more	86/80	135
CODESMITH-86 - debug	PC DOS	139
MacASM-full, fast, tools	MAC	100
MBP Cobol-86 - fast	8086	695
METAWINDOW - graph, fonts, clip	PC DOS	135
Micro: SubMATH-FORTRAN full	86/80	245
Microsoft MASM-86	MSDOS	85
MSD Debugger	MSDOS	119
Multilink - Multitasking	PC DOS	265
PC FORTH - + - well liked	MSDOS	219
PFIX-86 Debugger	MSDOS	169
PL 1-86	8086	495
Polylibrarian - thorough	MSDOS	95
PolyMAKE	PC DOS	95
PROFILER - flexible	MSDOS	125
Prolog-86-Learn, Experiment	MSDOS	125
SLK F - Copy Protection	PC DOS	150
SYMD debugger-symbols	PC DOS	119
TRACE86 debugger ASM	MSDOS	115

Note: All prices subject to change without notice. Mention this ad. Some prices are specials. Ask about COD and PDs. All formats available.

Call for a catalog, literature, and solid value

800-421-8006

THE PROGRAMMER'S SHOP™

128-I Rockland Street, Hanover, MA 02339
Visa Mass: 800-442-8070 or 617-826-7531 MasterCard

addset(x [] [x]).

i.e. adding x to the empty set produces the set composed of x ;

addset(x [x|t] [x|t]).

i.e. adding x to the set already headed by x produces no change;

addset(x [h|t] [h|t and x]) if
not-same(x h) and
addset(x t and x).

i.e. adding x to a set not headed by x is done by adding x to the tail of the set.

The clause not-same($a b$), used in addset and remove, is one that the underlying system should define. It is extremely difficult to describe the not-equal relation in Prolog.

We can implement—or describe—binary trees in terms of nested lists. In fact, their Prolog description is simply a rephrasing of the theoretical definition of a binary tree. Let's say that an empty list represents the empty tree, while a three-item list [lsub item rsub] represents the nonempty tree. item is the value at the root of the tree, and lsub and rsub are the left and right subtrees. Now we can define the relation addtree(x in out) such that out is the binary tree in with item x added to it:

addtree(x [] [] [x]).

i.e. adding x to an empty tree produces the tree with value x and empty subtrees;

addtree(x [l x r] [l x r]).

i.e. x is already in the tree that has it as a value;

addtree(x [l v r] [l new v r]) if
less(x v) and
addtree(x l new).

i.e. x goes into the left subtree of a tree that has a greater value;

addtree(x [l v r] [l v r new]) if
less(v x) and
addtree(x r new).

i.e. x goes in the right subtree of one with a lesser value. Lovely, isn't it? No doubt you can already visualize the Prolog description of a tree search, the relation intree(x tree) that is true if x appears at some level of tree.

Let us return to the domain of lists and sets long enough to describe the QuickSort algorithm: the relation qsort(in out) such that out is the list in with its items in sorted order. First, we enumerate enough of the limiting cases to cut the recursion off at the two-item list:

qsort([] []).

In that matching process lies much of the magic of Prolog. Many texts laboriously describe how variables are "instantiated" (I prefer to say "bound") as they are matched to constants, but the process is simply *copying with substitution*: the interpreter copies out the matched rule, substituting new values wherever a variable was matched.

The interpreter commences by comparing the first (and only) clause of the query to the first sentence in the data base:

length([a q j] what)
length([] 0)

They don't match; the mismatch comes just after the opening bracket when the constant a doesn't match the closing bracket. The interpreter tries again at the next sentence:

length([a q j] what)
length([head|tail] len) . . .

This succeeds. A new copy of the rule is made with the following substitutions:

a for head
[q j] for tail
what for len

The copied conjunction, with substituted values of the bound variables, reads:

(2) length([q j] tlen₂) and
sum(tlen₂ 1 what)?

The subscript in tlen₂ indicates that the variable tlen is unique and local to this query.

The first clause of the new query matches the second sentence in the data base, with the following bindings:

q for head
[j] for tail
tlen₂ for len

The conjunction is copied out once more, with substitution, to produce yet another nested query:

(3) length([j] tlen₃) and
sum(tlen₃ 1 tlen₂)?

Its first clause again matches the second sentence, this time binding the following:

j for head
[] for tail
tlen₃ for len

Its conjunction is copied out; after substitution, it yields:

(4) length([] tlen₄) and
sum(tlen₄ 1 tlen₃)?

Add EDITING to your Software with CSE Run-Time™

Your program can include all or a portion of the *C Screen Editor (CSE)*.

CSE includes all of the basics of full screen editing plus source in C for only \$75. For only \$100 more get CSE Run-Time to cover the first 50 copies that you distribute.

Use capabilities like Full cursor control, block move, insert, search/replace or others. Portability is high for OSes, terminals, and source code.

Call for the "CSE Technical Description" and for licensing terms and restrictions.

Full Refund if
not satisfied in
first 30 days.

Call 800-821-2492

**Solution
Systems**

335-D Washington Street
Norwell, MA 02061
617-659-1571

Circle no. 75 on reader service card.

Program Editing Breakthrough!

Save time. Stop worrying.

Use BRIEF: The most powerful UNDO in the industry lets you recover from any mistake. The most powerful editor combines just about every feature you have ever seen in a coherent, logical, practical set of commands. And the macro programming language can be used to change just about anything — to suit your style or background.

Try BRIEF — at no risk. With windows, UNDO of all operations, multifile editing and more, you could save time every day.

- Full UNDO (N Times)
- Edit Multiple Large Files
- True Automatic Indent for C
- Exit to DOS Inside BRIEF
- Uses All Available Memory
- Intuitive Commands
- Tutorial
- Keystroke Macros
- Windows (Tiled and "Pop Up")
- Unlimited File Size
- Reconfigurable Keyboard
- Online Help
- Search for "regular expressions"
- Mnemonic Key Assignments
- Horizontal Scrolling
- Comprehensive Error Recovery

PLUS a Complete, Powerful, Readable, Compiled MACRO Language

Availability: PC DOS-compatible systems. **Price:** Only \$195.

Win \$1,000 and recognition for the Outstanding Practical BRIEF Macro. Other awards to be given.

Try BRIEF. Use the Demo... or the full product

for 30 days. Call or write us... **800-821-2492**

**Solution
Systems**

BRIEF is a trademark of UnderWare.

Solution Systems is a trademark of Solution Systems.

335-D Washington St., Norwell, MA 02061 617-659-1571

Circle no. 93 on reader service card.

C Helper™

FIRST-AID FOR C PROGRAMS

Save time and frustration when analyzing and manipulating C programs. Use C HELPER's UNIX-like utilities which include:

- DIFF** and **CMP** — for "intelligent" file comparisons.
- XREF** — cross references variables by function and line.
- C Flow Chart** — shows what functions call each other.
- C Beautifier** — make source more regular and readable.
- GREP** — search for sophisticated patterns in text.

There are several other utilities that help with converting from one C compiler to another and with printing programs.

C Helper is written in portable C and includes both full source code and executable files for \$135 for MS-DOS, CPM-80 or CPM-86. Use VISA, Master Card or COD.

Call: **800-821-2492**

**Solution
Systems**

335-D Washington Street
Norwell, MA 02061
617-659-1571

Circle no. 94 on reader service card.

PROLOG-86™

Become Familiar in One Evening

Thorough tutorials are designed to help learn the PROLOG language quickly. The interactive PROLOG-86 Interpreter gives immediate feedback. In a few hours you will begin to feel comfortable with it. In a few days you are likely to know enough to modify some of the more sophisticated sample programs.

Sample Programs are Included like:

- an **EXPERT SYSTEM**
- a **NATURAL LANGUAGE INTERFACE**
(it generates a dBASE II "DISPLAY" command)
- a **GAME** (it takes less than 1 page of PROLOG-86)

PROTOTYPE Ideas and Applications QUICKLY

1 or 2 pages of PROLOG is often equivalent to 10 or 15 pages in "C" or PASCAL. It is a different way of thinking.

Describe the **FACTS** and **RULES** without concern for what the computer will have to do. Maybe you will rewrite in another programming language when you are done.

Programming Experience is not required but a logical mind is: PROLOG-86 supports the de facto STANDARD established in "Programming in Prolog."

CONTEST: Win \$1,000. Ask about it. Deadline of 4/30/85.

AVAILABILITY: PROLOG-86 runs on MSDOS, PC DOS or CPM-86 machines. We provide most formats. The price of PROLOG-86 is **only \$125**.

Full Refund if not
satisfied during
first 30 days.
800-821-2492

**Solution
Systems**

335-D Washington Street
Norwell, MA 02061
617-659-1571

Circle no. 95 on reader service card.

i.e. the empty list is sorted;

```
qsort([x ][x ]).
```

i.e. the one-item list is sorted;

```
qsort([a z ][a z ]) if
  leq(a z ).
qsort([a z ][z a ]) if
  less(z a ).
```

i.e. two-item lists are sorted by exchanging their items (we assume that the `leq()` and `less()` clauses are system defined).

Now we can describe the general solution of QuickSort: the items of the input list are partitioned into two lists, which are sorted separately:

```
qsort([a b c | t ] out ) if
  part([a b c | t ] l r ) and
  qsort(l sl ) and
  qsort(r sr ) and
  catenate(sl sr out ).
```

Assuming that the `catenate` relation already has been described, we need only to define the partitioning scheme, `part(in l r)`, such that *l* and *r* are lists containing all the items of *in* between them with every item in *l* being less than any item in *r*.

I've worked out two ways to describe the `part` relation. The first makes two passes over the data:

```
part(in l r ) if
  pivot(in p ) and
  partle(in p l ) and
  partge(in p r ).
```

Leaving aside the `pivot` relation, which describes *p* as a pivot value based on list *in*, we need definitions of `partle` and `partge`. The `partle` relation selects those elements of *in* that are less than or equal to *p*:

```
partle([ ] p [ ]).
partle([h | t ] p [h | z ]) if
  leq(h p ) and
  partle(t p z ).
partle([h | t ] p z ) if
  less(p h ) and
  partle(t p z ).
```

The `partge` relation is similar but, of course, selects only items that are greater than *p*.

It seemed somewhat inefficient to pass twice over the input list when dividing it, so I worked out an alternate scheme:

```
part(in l r ) if
  pivot(in p ) and
  partem(p in l r ).

partem(p [ ] [ ] [ ] ).
```

The first clause of this conjunction matches the first sentence:

```
length([ ] tlen4) . . .
length([ ] 0).
```

As a result, the variable *tlen₄* is bound to the literal number 0; it is effectively rewritten to 0 wherever it appears in the pending queries. Having verified the first clause of query (4), the interpreter considers its second clause, which now reads:

```
. . . sum(0 1 tlen3)
```

Assume that `sum` is a built-in relation; the system adds the known arguments and binds their sum to the unknown one, causing *tlen₃* to be rewritten as 1 wherever it appears. Query (4) is now complete; the system may return to consideration of query (3).

The second clause of query (3) now reads:

```
. . . sum(1 1 tlen2)
```

After that clause has been evaluated, references to *tlen₂* will read 2. That completes processing of query (3); the interpreter may return to query (2).

Its second clause now reads:

```
. . . sum(2 1 what )
```

Its evaluation rewrites all appearances of *what* as 3. The text of query (1) now reads:

```
length([a q j] 3).
```

Because this user-entered query has now been verified as truthful, the system can notify the user "yes" and may also display *what* = 3. That's not essential; a very simple Prolog might require the user to request a display explicitly, using a built-in relation such as:

```
length([a q j] what ) and
display(what )?
```

Had that been the case, verification of the first clause would have resulted in rewriting the query as:

```
length([a q j] 3) and
display(3)?
```

Nothing special has to be done by the system—the display argument falls out naturally from the implicit rewriting of variables as they are bound.

Thus, relatively simple matching and rewriting rules, applied recursively, produce effects all out of proportion to their size.

Cutting and Failing

Let us consider taking our member relation, defined by the following sentences:

B. G. MICRO

P. O. Box 280298 Dallas, Texas 75228

(214) 271-5546



74LS

LS00	.20	LS162	.65
LS01	.20	LS163	.45
LS02	.20	LS164	.65
LS03	.20	LS165	.90
LS04	.20	LS166	.99
LS05	.20	LS169	1.25
LS08	.20	LS174	.45
LS09	.20	LS175	.40
LS10	.20	LS181	1.50
LS11	.20	LS191	.90
LS12	.35	LS192	.80
LS13	.40	LS193	.80
LS14	.40	LS194	.65
LS15	.32	LS195	.60
LS20	.20	LS196	.70
LS21	.25	LS197	.85
LS27	.28	LS221	.65
LS30	.20	LS240	1.00
LS32	.25	LS241	.80
LS33	.40	LS242	1.00
LS37	.33	LS243	1.00
LS38	.30	LS244	1.00
LS42	.40	LS245	1.00
LS51	.24	LS251	.50
LS54	.25	LS253	.50
LS55	.24	LS257	.50
LS73	.35	LS258	.55
LS74	.30	LS259	2.00
LS85	.60	LS260	.50
LS86	.25	LS266	.40
LS90	.50	LS273	1.00
LS93	.55	LS279	.40
LS97	2.00	LS280	1.00
LS107	.37	LS283	.50
LS109	.25	LS290	.85
LS112	.30	LS293	.85
LS122	.45	LS298	.75
LS123	.55	LS299	1.60
LS124	2.75	LS323	2.60
LS125	.40	LS348	.75
LS126	.49	LS364	1.10
LS132	.50	LS366	.45
LS133	.35	LS367	.50
LS138	.45	LS368	.40
LS139	.40	LS373	1.00
LS151	.45	LS374	1.00
LS153	.50	LS375	.50
LS154	1.20	LS377	1.00
LS155	.50	LS378	.85
LS156	.50	LS390	1.00
LS157	.40	LS393	1.00
LS158	.50	LS399	1.25
LS160	.65	LS670	1.50
LS161	.50	25LS2569	3.00

TTL

7400	.22	7472	.25	74161	.60
7402	.22	7473	.34	74162	.60
7405	.22	7474	.40	74163	.60
7406	.33	7483	.45	74164	.80
7407	.33	7485	.55	74165	.80
7408	.24	7486	.30	74166	1.00
7410	.19	7490	.35	74173	.75
7413	.33	7493	.33	74174	.85
7420	.22	7496	.40	74175	.80
7425	.25	74107	.28	74181	1.50
7427	.25	74109	.45	74185	1.70
7428	.25	74121	.27	74192	.70
7430	.20	74123	.40	74193	.80
7432	.27	74125	.49	74195	.80
7433	.25	74132	.40	74199	1.00
7440	.19	74148	1.00	74221	1.00
7442	.40	74151	.50	74273	1.75
7448	.65	74153	.50	74365	.60
7451	.20	74154	1.19	74366	.60
7453	.22	74157	.50	74367	.60
7454	.22	74160	.80	74390	1.40

EPROM

2708 1KX8 450 n.s.	2.20
2758 1KX8 +5V 450 n.s.	2.00
2716 2KX8 450 n.s.	3.20
2716-1 2KX8 350 n.s.	4.95
2732 4KX8 450 n.s.	4.00
2732A-3	5.70
2732A-35	5.20
2532 4KX8 450 n.s.	3.00
2764-25	6.00
2764-3	5.00
27128-25	12.50
27128-3	11.50
27128-45	10.00

EPROM SPECIAL

We bought a large quantity of 2708s from a computer manufacturer who redesigned their boards. We removed them from sockets, erased and verified them, and now we offer the savings to you. Complete satisfaction guaranteed.

2708
\$1.49 or 10/\$12.00

STATIC RAM

2016-2KX8 200 n.s.	8/24.95
2101-1 - 256X4 500 n.s.	.75
21L02-1 350 n.s.	.65
2102AL-4 L.P. 450 n.s.	.49
2111-1 256X4 500 n.s.	2.00
2114L-3 1KX4 300 n.s.	1.50
	8/10.00
2125A-2 1KX1 70 n.s.	2.20
2142-3 1KX4 300 n.s.	1.50
TMS4044 (MCM6641 C-25)	
4KX1 250 n.s.	8/6.00
6116P-4-2KX8	
200 n.s. CMOS	8/27.50
6501-5 256X4 - CMOS	1.00

4K STATIC RAMS

LESS THAN 50c EACH
MK4104J-4 - 250 N.S. 18 Pin
Ceramic Computer Mfg.
Surplus. PRIME. Fully Static.
Easy to Use. Has Same Pin Out
as TMS4044, but slightly
different timing. With Specs.
(Mostek)

8 for 5.00 32 for 15.95
VERY LOW POWER!

DYNAMIC RAM

2108-4 8KX1	1.50
2118-4 16KX1-5Volt	1.50
4027-4KX1-250 n.s.	.80
4116-16KX1-250 n.s.	8/8.00
4116-16KX1-200 n.s.	8/11.50
4116 16KX1-150 n.s.	8/12.95
4164- +5v 64K 200 n.s.	8/32.00
4164 150 n.s.	8/34.00
TMS4416-16KX4-150 n.s.	5.25
MK4516-15 16KX1-5Volt	1.50
5280N-5 (2107B-4 + TMS4060)	
4KX1	8/3.95
41256 150 n.s.	8/165.00

SPECIAL

AY3-8910
W/60 Page Manual
New Price — \$7.00

5832 Clock-Calendar 2.95

8000

Z8002	20.00	8085	5.95
8035	4.50	8086-2	24.95
8039	4.50	8087-3	159.00
8080A	1.25	8088	15.00

8200

8202A	10.00	8255-5	5.00
D8203-1	29.95	8257	6.00
8212	1.50	8259A	3.50
8214	2.00	8259C-5	5.00
8216	1.75	8275	19.95
8228	3.25	8284	3.20
8237-5	7.50	8287	5.75
8250B	9.95	8288	7.50
8251	4.20		

Z80

Z80 2.5 MHZ CPU	1.25
Z80CTC	1.25
Z80DMA-DMA	7.50
Z80PIO	1.49
Z80SIO/O	3.99
Z80A-4MHZ CPU	1.99
Z80A-CTC	1.99
Z80A DART	7.50
Z80A-DMA	9.95
Z80A-PIO	1.99
Z80A SIO/O	5.99
Z80B 6 MHZ CPU	9.50

F.D. CONTROLLERS

1771 Single Density	12.95
1791 Double Density	20.00
1793	17.50
1797	17.50
2793	22.00
2797	20.00

CONTROLLER SET

THREE CHIP SET

1797 or 1793. 2143-03. 1691

by W.D. Compare at up to 86.85.

B.G. SPECIAL

All 3 for only \$22.95

UART

TR1602B (COM 2017)	1.75
IM6402-(1863)+5v High speed	
AY5-1013 pin out	2.95
INS 8250B	9.95

6800

6800	2.50	6840	10.00
6802	3.50	6845P	7.50
6803	5.00	6845S	7.50
6809EP	3.99	6850	2.60
6810	2.00	68A09EP	4.99
6820	3.25	68A21	3.00
6821	2.00	68B45	10.00

6500

6502	2.60	6545	5.00
6522	6.95	6551	5.00

SPECIALS

1408L8	2.00
BR1941L	5.95
LM311	.40
LF353	.60
LF356	.60
TL494	2.50
LM555	.30
733	.60
9401	5.00
DS8835	1.50
MC4024P	2.50
NE592	1.50
LM319	1.00
MC1350	1.00
LM2917	1.50
LM339	.50

SOCKETS

Low Profile SOLDER TAIL

6 Pin	14/1.00
8 Pin	13/1.00
14 Pin	10/1.00
16 Pin	8/1.00
18 Pin	8/1.00
20 Pin	7/1.00
22 Pin	7/1.00
24 Pin	6/1.00
28 Pin	6/1.00
40 Pin	5/1.00

BUY \$10

GET \$1.00 - FREE CHOICE

2114 SPECIAL!

COMPUTER
MANUFACTURERS
EXCESS INVENTORY
SALE!

PRIME! 2114-300 n.s.
INCREDIBLE PRICE!

YOU SAVE!

8/\$8.00

GUARANTEED

CRYSTALS

32.768 KHz SPECIAL	.65
262.144	.75
300.000	1.00
307.200	1.25
1.5432 Mhz	.75
1.8432	2.49
2.000000	2.49
2.560	1.49
3.000	1.15
3.120	1.20
3.2	1.49
3.4560	1.49
3.579545	.75
4.000	2.49
4.194304	1.50
4.433618	.75
4444.000	1.25
4.9152	2.49
4.916 Bd. Rate	1.25
5.000000	1.50
5.0688	3.75
5.616	1.59
6.000	2.49
6.176	1.49
7.164112	1.00
7.3728	1.49
8.000	1.49
9.000	1.49
9.900000	1.25
10.69425	3.75
10.8864	1.49
10.920	1.49
11.088	1.59
12.000	2.75
13.440	1.00
14.31818	2.00
15.2	1.10
16.00000	1.50
16.5888	1.49
17.430	2.49
18.000	1.49
18.2259	1.00
18.4320	1.49
20.000	3.75
21.87108	1.00
22.092	1.00
32.000	2.49
40.000	2.00
87.3333	1.00
91.000	1.00
104.8	1.00

TERMS: (Unless specified elsewhere) Add \$1.50 postage, we pay balance. Orders over \$50.00 add 85c for insurance. No C.O.D. Texas Res. add 6-1/8% Tax. 90 Day Money Back Guarantee on all items. All items subject to prior sale. Prices subject to change without notice. Foreign order - US funds only. We cannot ship to Mexico. Countries other than Canada, add \$3.50 shipping and handling.


```

partem(p [h|t] [h|l] r) if
  leq(h p) and
  partem(p t l r).
partem(p [h|t] l [h|r] r) if
  less(p h) and
  partem(p t l r).

```

Finally, we may define the pivot relation in a variety of ways. The simplest is to use the first item: `pivot([h|t] h)`. That causes poor performance, however, when the input is already sorted. Because, by our definition of the general case of `qsort`, we need to describe a pivot only for lists of three or more elements, we get a better average partition with a pivot that is the mean of the first and third items:

```

pivot([a b c|t] p) if
  SUM(a c s) and
  PROD(2 p s 0).

```

I've tested these versions of QuickSort on Micro-PROLOG, although not on lists long enough to compare their relative speeds.

Storing Data

But, I hear you asking, how can we conduct a meaningful test of a sort if the only sample data is what we can type in a query? It may be amusing to have a dialogue like

```

qsort([3 2 1] what)?
yes, what = [1 2 3].

```

but this hardly amounts to useful data processing. Where do we put stuff so we can work on it?

That appears to be a natural question, but it reveals unthinking prejudices.

Procedural languages enforce a radical distinction between code and data. Data is static; it is stored in variables; it is acted upon by code. Code is dynamic; it is expressed by statements; it takes data as input and produces data as output. These notions are so engrained in the conventional programming languages that we aren't even aware of them—until we encounter a notation like Prolog, which denies them all!

Prolog makes no difference between data and code; the distinction is simply not relevant. But, you ask, how do we store information? We do it by *asserting the truth of that information*. The asserted sentences become part of the data base. Everything is sentences; we may think of *any* sentence as either code or data. Consider that the query

```

qsort([3 2 1] what)?
yes, what = [1 2 3].

```

could be answered equally well by our algorithmic `qsort` definition or by a simple assertion of fact: `qsort([3 2 1] [1 2 3])`. In principle, there is *no difference* between the algorithmic `qsort` (code) and the `qsort` implemented by an infinite set of facts (data).

But practical problems arise from the mode of interactive use. How can we develop data during one query, yet have the same data available for another later query? Suppose, for

```

member(item [item|tail]).
member(item [head|tail]) if
  member(item tail).

```

and apply to it the query

```
member(j [j k])?
```

These sentences are not independent; the query matches both of them. It matches

```
member(item [item|tail]).
```

with *item* bound to *j* and *tail* bound to *[k]*, and it also matches

```
member(item [head|tail])...
```

with *item* bound to *j*, *head* to *j*, and *tail* to *[k]*. Which sentence would Prolog choose to use, and what would happen then?

The Prologs that run on personal computers practically guarantee that the sentences of a relation will be tested in the order they were entered to the data base. If the limiting case is entered first, as here, it will be matched first. You'll find plenty of cases where programmers rely on this implicit order of evaluation. If the interpreter is asked to find only one answer to the query, it will match to the first sentence (a fact) and, having verified the query, will reply "yes" and stop.

But in some cases, the interpreter will continue to look for any further answers to a query. In our example, it would continue on to match the query to the second sentence. That sentence is a rule, and its conjunction specifies testing the query, rewritten as follows:

```
member(j [k])?
```

This does not match the first sentence, but it does match the second one, binding *item* to *j*, *head* to *k*, and *tail* to the empty list. This results in a further nested query

```
member(j [ ])?
```

which matches neither sentence. Thus, this branch of the trail ends with failure. No additional answers have been found; the only effect has been some wasted processing time.

We could have avoided the extra work had we used the built-in "cut" relation in the first sentence. Cut has only the descriptive meaning of true, so inserting "cut() and ..." in any conjunction has no logical effect. However, this insertion has the powerful side effect of telling the interpreter to attempt no alternative sentences for this relation! To use it in our example, we would write the first sentence as:

```
member(item [item|tail]) if
  cut( ).
```

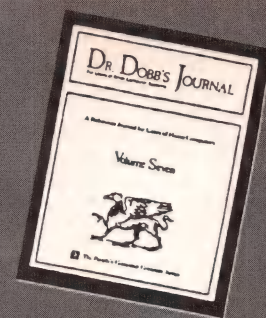
(The clause `cut()` is written as `(/)` in Micro-PROLOG and !

Announcing BOUND VOLUME 7

Every 1982 Issue Available For Your Personal Reference.

Vol. 7 1982

In 1982 we introduced several significant pieces of software, including the RED text editor and the Runic extensible compiler, and we continued to publish utility programs and useful algorithms. Two new columns, The CP/M Exchange and The 16-Bit Software Toolbox, were launched, and we devoted special issues to FORTH and telecommunications. Resident Intern Dave Cortesi supplied a year of "Clinic" columns while delivering his famous review of JRT Pascal and writing the first serious technical comparison of CP/M-86 and MSDOS. This was also the year we began looking forward to today's generation of microprocessors and operating systems, publishing software for the Motorola 68000 and the Zilog Z8000 as well as Unix code. And in December, we looked beyond, in the provocative essay, "Fifth-generation Computers."



Vol. 1 1976

The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering," front-panel, machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, home-brew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

Vol. 2 1977

1977 found DDJ still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic, and NIBL.

Articles are about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

Vol. 3 1978

The microcomputer industry entered its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this, they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.

Languages covered in depth were SAM76, Pilot, Pascal, and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors, and much, much more.

Vol. 4 1979

This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use the processors which have proven longevity—primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit Conversion, Pseudo-random Sequences, and Interfacing a Micro to a Mainframe—more than ever!

Vol. 5 1980

All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (DDJ's all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and DDJ published Ron Cain's immediately famous Small-C compiler—reprinted here in full! Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Run-time Library for the Small-C Compiler and, as always, even more!

Vol. 6 1981

1981 saw our first all-FORTH issue (now sold out), along with continuing coverage of CP/M, small-C, telecommunications, and new languages. Dave Cortesi opened "Dr. Dobb's Clinic" in 1981, beginning one of the magazine's most popular features.

Highlights: information on PCNET, the Conference Tree, and The Electric Phone Book, writing your own compiler, a systems programming language, and Tiny BASIC for the 6809.

YES! ☐ Please send me the following Volumes of **Dr. Dobb's Journal**.
☐ ALL 7 for ONLY \$165, a savings of over 15%!

Payment must accompany your order.

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

Mail to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303
Allow 6-9 weeks for delivery.

This offer expires February 28, 1985

Vol. 1	x	\$26.75	=	_____
Vol. 2	x	\$27.75	=	_____
Vol. 3	x	\$27.75	=	_____
Vol. 4	x	\$27.75	=	_____
Vol. 5	x	\$27.75	=	_____
Vol. 6	x	\$27.75	=	_____
Vol. 7	x	\$30.75	=	_____
All 7	x	\$165.00	=	_____

Sub-total \$ _____

**Postage & Handling
Must be included with order.**

Please add \$1.25 per book in U.S.
(\$3.25 each surface mail
outside U.S. Foreign Airmail
rates available on request.)

TOTAL \$ _____

instance, that we want to test our addset relation, retaining its output to use as input to the next test.

We start by asserting that one particular set—which we name with an unusual constant—exists and is null:

“The Set”([]).

We can query it just like any other relation:

“The Set”(what)?
yes, what = [].

More importantly, however, we can query it as part of a test program, test-add(*x*), that describes the effect of testing addset with item *x*:

```
test-add(x) if
  “The Set”(in ) and
  addset(x in out ) and
  display(out ) and
  retract([“The Set” in ]) and
  assert([“The Set” out ]).
```

With this program, we begin using Prolog syntax in a linear, time-bound way. The display, retract, and assert relations all are system-defined. Each has a trivial descriptive sense and an important side effect on the Prolog environment.

Descriptively, the display(*x*) relation is simply true for any item *x*. Therefore, “display(*item*) and...” may be inserted in any conjunction. Its side effect is to display *x* at the console (the relation is called PP in Micro-PROLOG).

The relation retract(*list*) is true when *list* describes a sentence presently in the Prolog data base. Its side effect is to delete that sentence.

The relation assert(*list*) is true when *list* describes a valid Prolog sentence. Its side effect is to assert that new sentence, making it part of the data base. The format for describing a sentence as a list is system-dependent.

Thus, the sense of the test-add relation is: get the current value of “The Set”; add item *x* to it and display the result; replace the definition of “The Set” with the augmented set. It permits dialogues like this:

```
test-add(5)?
[5]
yes.
test-add(fred)?
[5 fred]
yes.
“The Set”(what )?
yes, what = [5 fred].
```

A test-add query is clearly time dependent. It makes no sense unless the retraction is done *after* addset and *before* assert.

This is not elegant.

This is, in fact, an indefensible kludge, so I won't try to defend it. It is how the concept of the global variable is introduced into Prolog—and how we can simulate almost any sort of data storage mechanism. For example, extending the method of test-add to make a test bed for binary tree algo-

in Edinburgh-style systems.) The effect, upon a successful match to this rule, would be to cut off any attempt to find an alternate sentence for member. The second sentence would be removed from consideration as soon as the first was matched. Only if the first sentence failed to match could the second be applied.

This interpretation of cut, of course, assumes that the interpreter processes sentences sequentially in the order they are asserted. If it does not, it might attempt the second sentence before the first one, and adding cut to the first accomplishes nothing.

Consider the execution of the original member relation, without cut, in a machine that has And-parallel hardware (see the sidebar “Prolog In Parallel”). The system can execute tests of both sentences of the member relation at once, and it will find that both match the query. Now what? Well, the machine might favor facts over rules, which would make it select the right path. Or it might keep an entry sequence number with each sentence and favor the one with the lower sequence number. Or it might simply follow up both paths in parallel. In the last case, the first sentence, being a fact, will terminate quickly with success, while the second will end in failure one nesting level later.

In an And-parallel machine, the effect of the cut relation is nondeterministic. One can imagine some kind of hardware feature in which execution of cut commits intersentence fratricide, the cutting sentence aborting parallel execution of any other sentence from the same relation, but would that be useful? We could not tell how far another execution unit might have gone, executing a different sentence of the relation, before the one execution unit reached the cut-clause. So programmers using cut to avoid accidental, multiple uses of a file-read or some other do-once clause—as is recommended in Prolog tutorials—should reflect that this program is not suitable for And-parallel execution.

When you are designing Prolog solutions, it is always best to describe a relation in sentences that are independent of each other in all cases, without regard to execution sequence. We could accomplish that in our example using a not-same relation:

```
member(item [item|tail]).
member(item [head|tail]) if
  not-same(item head ) and
  member(item tail ).
```

These sentences are complete and independent; the solution has no dependency on either order of execution or degree of parallelism.

The “fail” relation has the descriptive meaning “false—this sentence does not match.” It has some uses, but it is hard to find examples that do not rely on order of execution.

One possible use of fail is to implement a not-same relation:

```
not-same(item item ) if
  cut( ) and fail( ).
not-same(x y ).
```


This is sequence dependent, as you might expect from the appearance of cut(). If a not-same clause is being tested with arguments that are the same, the clause will match to the first sentence. The cut clause ensures that the interpreter will attempt no further sentences of not-same; the fail clause ensures that the query fails—the same items are not not-same. A query of not-same with arguments that are not the same will not match to the first sentence, so the interpreter will test it (successfully) against the second one. That sentence is a fact, so not-same is verified.

As with most definitions that use cut(), this is utterly without meaning unless the order of evaluation of sentences is certain; thus, it is unusable in an And-parallel machine.

rithms (using a global “The Tree”) is simple. More complicated schemes are also possible: we can use assert and retract to implement (*not* describe!) a sparse array. (The array is a relation with its elements stored as facts of the form array-*(sub value)*, meaning the array element at position *sub* has this *value*.)

The Meta-Variable

Some Prologs permit a variable to appear in the position of the relation-name in a clause. This feature, the meta-variable, makes it possible to write Prolog algorithms *about* Prolog.

As an example of how we might use the meta-variable, consider the “over” notation of APL. In that language, a function may be applied “over” an array. For instance, in

+ / 3 1 9 6 4

the plus is the function being applied, and the slash means apply it “over” the following list of numbers. The effect of “over” is to insert the function symbol between every pair of numbers, so that the preceding line is equivalent to:

3 + 1 + 9 + 6 + 4

In short, “plus over” is equivalent to summation, “max over” to largest, and so on.

We can write an “over” relation in Prolog if we use the meta-variable. The relation would be *over(fun list out)* with *out* as the result of applying *fun* between every pair of items in *list*. A relation named *fun* must exist and have the form *fun(a b z)*, meaning *a fun b* yields *z*. In addition, we need a sentence, *identity(fun id)*, such that *id* is the “identity element” of the function: the value is such that *fun(a id a)*.

Here is a collection of sample functions:

```
plus(a b z) if
  SUM(a b z).
identity(plus 0).
times(a b z) if
  PROD(a b z 0).
identity(times 1).
max(a b a) if
  leq(b a).
max(a b b) if
  less(a b).
```

identity(max 9E99)

And here is the definition of “over”:

```
over(fun [ ] id) if
  identity(fun id).
```

```
over(fun [a|t] z) if
  over(fun t b) and
  fun(a b z).
```

This is quite faithful to the APL model, even executing right-to-left through the items and returning the identity element when applied to an empty array.

Summary

Prolog is a notation that allows us to express many algorithms in neat and elegant ways and that forces others into distorted shapes (as Bob Morein puts it, if it’s hard in Fortran, it’s easy in Prolog—and vice versa!). Nevertheless, it is educational to rewrite familiar algorithms in a new notation.

But the greatest value of Prolog is that, in the process of understanding it, a programmer must alter and reexamine the fundamental assumptions of his or her craft. Prolog demonstrates that the distinction between code and data is artificial and unnecessary, and it exposes the many assumptions we make about the flow of time. Prolog or a notation descended from it may or may not become important in the fifth and further generations of computing; it doesn’t matter. What matters is that, after learning it, you will think about programming in a whole new way.

References

1. *Programming in Prolog*, W. F. Clocksin and C. F. Melish, Springer-Verlag, New York 1981.
2. Micro-PROLOG is a product of Logic Programming Associates, 10 Burntwood Close, SW 18 3JU, London, England.
3. Documented in *User's Guide to DECsystem-10 Prolog*, L. M. Pereira, F. Pereira, and D. Warren, and in *A Guide to Version 3 of DEC-10 Prolog*, L. Byrd, F. Pereira, and D. Warren. Both papers are published through the Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 193.

Tax Advisor A Prolog Program Analyzing Income Tax Issues

by Dean Schlobohm

The winner of our AI programming contest is a program that knows how it knows what it knows, and why it wants to know what it doesn't know.

A year ago, in March 1984, we celebrated DDJ's new relationship with M&T Publishing by announcing an artificial intelligence programming contest. We would award a \$1000 prize, we said, to the entry that best implemented artificial intelligence techniques in a practical application on a microcomputer. We didn't say so, but we expected the "practical" criterion to direct people toward the "applied" end of AI work, and it did. Although we gave some guidelines (strive for memory efficiency, real-world usefulness, and advancing the state of the art), we left the judging criteria somewhat loose, in keeping with the nature of the definition of artificial intelligence. We didn't pretend to know exactly what we were looking for, but we figured that we'd know it when we saw it. And we did.

It could have been a can of worms. We didn't specify processor, language, operating system, disk format. We didn't even specify that it be for a disk-based machine! For all we knew, we might get a thousand different programs for a thousand different machines.

We weren't too worried, though. The prize was not extravagant for the amount of work that was required, and we expected to receive a small number of entries. And we did. After preliminary culling, we found ourselves with only four entries. For four machines and four disk formats. From Mexico came Gerardo Cisneros' program for solving certain kinds of algebraic problems that arise in quantum mechanics. We were awed by the idea of his crunching matrices of many-electron wavefunctions on a 56 TPA CP/M system. There was Richard Grigonis's Prison Doctor, which took the overused Doctor model of Weizenbaum's Eliza program and turned it into something ingenious and new. This was only the latest in a series of entries by Richard, one of our most creative contributors and the author of last year's controversial "Sixth Generation Computers." Stephan Heumann brought us Logicon, a Commodore 64 BASIC program that simplified Boolean expressions. Stephan's program combined the formalisms of G. Spencer-Brown with some insights of his own and made clever use of color to discriminate levels of analysis. Then there was Dean Schlobohm's tax advisor program, a solid and impressive little expert system. I'd add, "just in time for doing your taxes," but I doubt that many readers of DDJ are prepared to make use of this kind of legal tax advice.

Judging was no trivial matter, even with so few entries. We enlisted the aid of referees and friends with the necessary machines and appropriate expertise. We ran every program past an AI expert who judged whether it embodied the techniques of AI, a programmer competent to evaluate the quality of the code, someone conversant with the subject area of the program (logic and physics, anyway; we winged it a little regarding law and forensics), a tester who could evaluate whether and how well the program actually worked, and three editors. In the end, we chose a program that demonstrated clear understanding of what applied AI techniques are and how an AI program ought to behave. It also turned out to be a good introduction to programming in Prolog, and

Dean A. Schlobohm, MICRO-AI, P.O. Box 91, Rheem Valley, CA 94570. Copyright © 1984 Dean A. Schlobohm.

this issue consequently shaped itself into a Prolog issue. One criterion that emerged in the judging process was justification: an AI program ought to be able to tell you why it did what it did. The program we chose did that, too.

We may yet print some of the runners-up. But this month we are pleased to present the winner of our AI programming contest, Dean Schlobohm's Tax Advisor. —Ed.

The prototype computer program TA (Tax Advisor) assists attorneys in analyzing the constructive ownership of stock rules set forth in Section 318(a).¹ Basically, Section 318(a) sets forth a number of rules under which a given person (e.g., an individual, trust, estate, partnership, or corporation) will be deemed to own stock that is actually, or constructively by the application of Section 318(a), owned by another person. For example, Section 318(a)(1) provides that an individual is considered to own the stock actually owned by his or her spouse, children, grandchildren, and parents. Similarly, Section 318(a)(2)(A) provides that stock owned by a partnership is considered to be proportionally owned by its partners.

Although the rules set forth in Section 318(a) involve little uncertainty, their application to a given factual situation is at times tedious. Furthermore, because stock that is deemed to be owned by a given person as a result of the application of Section 318(a) may, under certain conditions, be reattributed to another person, the statute provides many opportunities for the attorney to make a mistake in his or her analysis. As a result, I have developed two prototype computer programs that perform the required legal analysis.

The computer programs are written in the language Prolog. Basically, Prolog permits rules of the form:

If: A is true; and
 B is true; and
 C is true.
Then: D is true.

to be represented as:

D :-
 A,B,C.

One version of TA consists of a set of Prolog rules that describe all of the legal rules of Section 318(a). For example, assume: (1) **spouse(H,W)** means that H is the spouse of W; (2) **owns(I,N,C)** means that I actually owns N shares of corporation C stock; and (3) **num-con-own(H,N,C)** means that H constructively owns, as a result of Section 318(a)(1)(A), N shares of corporation C stock.

Then the rule of Section 318(a)(1)(A) that an individual is deemed to own the stock owned by his or her spouse may be expressed by the Prolog clause:

num-con-own(H,N,C) :-
 spouse(H,W),
 owns(W,N,C).

A second version of TA includes the facility to "explain" its reasoning. For example, the program will ask the user

only for the information that it needs to reach its conclusion. If the user is unsure why the computer asks a given question, he or she can ask the computer **why** the information is needed. The computer will then set forth an explanation of its reasoning in terms of the rules that it has used up to the point the question is asked. Moreover, the user can ask the computer for **help** in response to a given question. For example, if the user enters **help** in response to the question "Does the client have any children?" the program will explain that "children" includes legally adopted children. After the program has reached its conclusion, the user can ask the computer **how** the program reached its result. This explanation will consist of the set of rules and facts that the computer used in reaching its conclusion.

Unlike other artificial intelligence programs in the legal area,² the major goal of TA has been to develop a program that could be used by a practicing attorney with little or no knowledge of computers. Thus, a substantial amount of time has gone into designing an interface so that an attorney could use TA without understanding the computer language in which it is written.

Both programs currently run on a Compupro 816A microcomputer under the CP/M-86³ operating system. Because the current version of the Prolog interpreter (Prolog-86⁴) in which the programs are written can access only 128K of RAM, the explanation features have only been included in a program that contains the family attribution rules of Section 318(a)(1). The program without the explanation features, however, has correctly analyzed the examples set forth in Treasury Regulations §§1.318-2, 3, and 4.

A Brief Review of Section 318(a)

Basically, Section 318(a) sets forth rules under which certain individuals and entities are considered to own corporate stock that is, in fact, actually owned by someone else. This is called "constructive ownership of stock" or "attribution of stock ownership." Section 318(a) creates four types of relationships that may create primary attribution of ownership. These four relationships are:

- (1) Attribution between members of a family;
- (2) Attribution from an entity (e.g., a partnership or corporation) to person holding a beneficial interest in the entity (e.g., a partner or shareholder);
- (3) Attribution from a person (e.g., a partner or shareholder) to an entity (e.g., a partnership or corporation);
- (4) Attribution to owners of options to acquire stock.

Each of these will now be discussed in order, followed by a discussion of reattribution.

Attribution Between Family Members

Section 318(a)(1) provides that an individual is considered as owning stock owned, directly or indirectly, by or for:

- (1) his or her spouse, other than a spouse who is legally separated from the individual under a decree of divorce or separate maintenance;
- (2) his or her children, grandchildren, and parents. For purposes of Section 318(a)(1), a legally adopted child of an individual is treated as a child of such individual.

It should be noted that although a grandparent is deemed

to own stock owned by his or her grandchildren, a grandchild is not deemed to own stock owned by his or her grandparents. Furthermore, an individual is not deemed to own stock owned by his or her brothers, sisters, aunts, uncles, and other relatives.

As an example of Section 318(a)(1), assume that H, an individual, his wife, W, his son, S, and his grandson (S' son), G, each own 25 outstanding shares of Corporation X. Then H, W, and S are each considered as owning 100 shares, the 25 shares they actually own and 75 shares from the other three individuals. G, however, is considered as owning only 50 shares because the shares of his grandparents, H and W, are not attributed to him.

Attribution from an Entity to a Beneficiary of That Entity

Attribution from Partnerships and Estates

Section 318(a)(2)(A) provides that stock owned, directly or indirectly, by a partnership or an estate is considered to be owned proportionately by its partners or beneficiaries.

For example, assume that A, an individual, owns a 50% interest in a partnership. Further, assume that this partnership owns 50 out of the 100 outstanding shares of Corporation X, the remaining 50 shares being owned by A. Then A is considered to be the owner of 75 shares, the 50 shares he or she actually owns plus 50% of the shares owned by the partnership.

Attribution from Trusts

Except for "grantor" trusts and "pension" trusts, stock owned, directly or indirectly, by a trust is considered as owned by its beneficiaries in proportion to the actuarial interest of the beneficiaries in the trust.

For example, assume that a testamentary trust owns 25 of the outstanding 100 shares of stock of Corporation X. Further, assume that A, an individual, holds a vested remainder in the trust having a value, computed actuarially, equal to 4% of the value of the trust property. Then A is considered as constructively owning 1 share (4% of 25).

In the case of a "grantor" trust (e.g., a revocable trust), stock owned, directly or indirectly, by the trust will be considered as being owned by the person who is treated as the owner of the trust pursuant to Sections 671-677.⁵

Attribution from Corporations

Section 318(a)(2)(C) provides that if 50% or more in value of the stock of a corporation is owned, directly or indirectly, by a person, such person is considered as owning the stock owned, directly or indirectly, by the corporation in that proportion which the value of the stock that such person so owns bears to the value of all of the stock of the corporation.

For example, assume A and B, unrelated individuals, own 70% and 30%, respectively, in the value of the stock of Corporation M. Assume further that Corporation M owns 50 of the 100 outstanding shares of stock of Corporation O, the remaining 50 shares being owned by A. Then A is considered as owning 85 shares, the 50 shares that he or she actually owns plus 70% of the 50 shares owned by Corporation M. Because B does not own 50% or more of Corporation M, he or she is deemed to own no shares in Corporation O.

Attribution to Entities

Attribution to Partnerships and Estates

Generally, Section 318(a)(3)(A) provides that stock owned by a partner or a beneficiary of an estate is considered as owned by the partnership or estate. It should be noted that all of the shares owned by a partner or a beneficiary of an estate are attributed to the partnership or estate, not just a partner's or beneficiary's pro rata share.

For example, assume that A, an individual, has a 50% interest in a partnership. Further, assume that A owns 50 shares of stock in Corporation X. Then the partnership is considered as owning the 50 shares of Corporation X stock owned by A.

Attribution to Trusts

Except in the case of "grantor" trusts and "pension" trusts, stock that is owned, directly or indirectly, by a beneficiary of a trust is considered as owned by the trust, unless the beneficiary's interest is considered a "remote contingent interest." A "contingent interest" of a beneficiary is considered "remote" if, under the maximum exercise of discretion by the trustee in favor of the beneficiary, the value of such interest, computed actuarially, is 5% or less of the value of the trust property.

For example, assume that A, an individual, holds a vested remainder in a testamentary trust having a value, computed actuarially, of 4% of the value of the trust property. Furthermore, assume that A owns 75 shares of stock in Corporation X. Then the testamentary trust is considered to own A's 75 shares of Corporation X because A's interest is vested and not contingent.

In the case of a "grantor" trust, all stock owned by the grantor is deemed to be owned by the trust.

Attribution to Corporations

If 50% or more in value of the stock in a corporation is owned, directly or indirectly, by any person, then the corporation is deemed to own the stock owned by that person. Thus, as in the case of attribution from partners and beneficiaries to partnerships and estates, respectively, the corporation is deemed to own all of the stock owned by the shareholder, not just a proportionate amount.

For example, assume that A owns 70% of the value of Corporation M. Assume that Corporation M and A each own 50 of the 100 outstanding shares of stock of Corporation O. Then Corporation M is considered as owning 100 shares of Corporation O, the 50 shares that it actually owns and the 50 shares that A owns.

Attribution as a Result of Options

Section 318(a)(4) provides that if any person has an option to acquire stock, such stock will be considered as owned by such person. For purposes of this Section, an option to acquire such an option and each one of a series of such options is considered as an option to acquire the stock.

For example, assume that A and B, unrelated individuals, own all of the 100 outstanding shares of Corporation X, each owning 50 shares. Assume further that A has an option to acquire 25 of B's shares and has an option to acquire a further option to acquire the remaining 25 of B's shares. Then A

is considered as owning the 100 shares of stock of Corporation X.

Rules of Reattribution

Section 318(a)(5) provides certain rules for "reattribution." Except as provided below, any stock that is deemed to be owned by a person as a result of one of the four attribution rules set forth above is considered to be actually owned by such person for purposes of reapplying the above four attribution rules.

For example, if a trust owns 50% of the stock of Corporation X, stock of Corporation Y owned by Corporation X is attributed to the trust and may then be further attributed to the beneficiaries of the trust.

There are, however, two major exceptions to this rule. First, stock constructively owned by an individual as a result of the application of the family attribution rules of Section 318(a)(1) is not treated as owned by such individual for the purpose of again applying the family application rules in order to make another individual the constructive owner of such stock. For example, assume H, an individual, owns 50 shares of stock in Corporation X. Further, assume that H's son, S, and daughter, D, each owns 50 shares in Corporation X. Then S is deemed to own only 100 shares in Corporation X, the 50 shares he actually owns plus the 50 shares actually owned by his father, H. He is not deemed to own the stock owned by his sister, D.

There is a further exception to the exception for reattribution of family ownership that permits reattribution, in effect, among family members. Section 318(a)(5)(D) provides that, when an individual is deemed to own stock by either applying the family attribution rules or the option rules, then he or she is considered to own the stock by reason of the option rules. In other words, the option rules take precedence over the family attribution rules.

Finally, Section 318(a)(5)(C) provides that stock constructively owned by a partnership, estate, trust, or corporation by reason of the attribution rules set forth in Section 318(a)(3) will not be considered as owned by the entity for purposes of applying the attribution rules set forth in Section 318(a)(2) in order to make another the constructive owner of such stock. For example, if two unrelated individuals are beneficiaries of the same trust, stock owned by one of the beneficiaries that is attributed to the trust under Section 318(a)(3) will not be reattributed from the trust to the other beneficiary. It should be noted, however, that stock constructively owned by reason of Section 318(a)(2) may be reattributed under Section 318(a)(3). Thus, for example, if all of the stock of Corporations X and Y is owned by A, stock of Corporation Z held by Corporation X is attributed to Corporation Y through A.

The Section 318(a) Rules in Prolog

This part of the paper will discuss the Prolog implementation of the rules set forth in Section 318(a) and the Treasury Regulations thereunder.⁶ The complete listing of the program is contained in Listing One (page 75). In Listing Two (page 79), several of the examples set forth in the Treasury Regulations under Section 318(a) are solved using TA.

The Interface Program—section318

The Prolog procedure **section318** controls the general interaction between the computer program and the user. After displaying an introductory message to the user, TA asks the user for the name of the client. This will be used throughout the program in communicating various information to, and asking various questions of, the user. For purposes of the remainder of this paper, the client's name will be **Client**.

TA then calls the procedure **type(Client)**. This procedure simply asks whether **Client** is an individual, partnership, estate, trust, grantor trust, or corporation. Once the user has told the computer what type of entity **Client** is, **type** then adds this information to the computer's data base. This information is later used to restrict the amount of searching that the computer must do to reach its conclusion. For example, as discussed below, if **Client** is not an individual, the program will not ask for the family members of **Client** and will automatically set the number of shares attributed to **Client** as a result of the family attribution rules set forth in Section 318(a)(1) to zero.

TA then asks the user for the name of the corporation whose stock may be attributed to **Client** under Section 318(a). For purposes of the remainder of this paper, the corporation in question will be called **Corp**.

After this is done, TA will attempt to find the number of shares of **Corp** stock that **Client** actually owns. The program does this by calling the procedure **num_a_own(Client, Corp, N_actual)**. This procedure will be discussed below.

After TA knows the number of shares of **Corp** stock that **Client** actually owns, it then calls the procedure **num_con_own(Client, Corp, N_con)**, which determines the number (**N_con**) of shares of stock of **Corp** that the **Client** constructively owns as a result of the rules set forth in Section 318(a). This procedure is the major procedure of TA and will be discussed in detail below.

Finally, **section318** prints out the number of shares of **Corp** stock that **Client** actually and constructively, by application of Section 318(a), owns. After this is done, TA asks the user if he or she wants to run the program again for a new client. If the answer is yes, the program will start over. If the user answers no, the program returns the user to the Prolog interpreter and awaits new commands.

The Prolog Procedure for Actual Ownership—**num_a_own**

The Prolog procedure **num_a_own** determines the number of shares of **Corp** stock that **Client** actually owns. The procedure first knows that a client will not actually own any stock in himself. Thus, the first clause of the procedure is simply:

```
num_a_own(Client, Client, 0) :- !.
```

If the client is not identical to the corporation, **num_a_own(Client, Corp, N_actual)** then asks the user for the number of shares of stock that **Client** actually owns in **Corp**. The program expects the user to enter an integer, and if he or she does not do so, it prints an error message and again asks the user for the number of shares of stock that **Client** actually owns.

Once TA knows how many shares of stock the client actually owns, it adds this fact to its data base.⁷

The Prolog Procedure for the Constructive Ownership Rule Set Forth in Section 318(a)— `num_con_own`

The procedure `num_con_own(Client, Corp, Number)` is defined as:

```
num_con_own(Client, Corp, Number) :-  
    num_family(Client, Corp, N1),  
    num_from(Client, Corp, N2, zzzz),  
    num_to(Client, Corp, N3),  
    num_options(Client, Corp, N4), !,  
    Number is N1 + N2 + N3 + N4.
```

The procedure simply calls four other procedures, `num_family`, `num_from`, `num_to`, and `num_options`. As discussed below, these four procedures find the number of shares of stock that **Client** constructively owns in **Corp** by the application of the rules of Sections 318(a)(1), (2), (3), and (4), respectively.

`num_con_own` then simply adds the number of shares that **Client** constructively owns as a result of each of the four constructive ownership rules to find the total number of shares that **Client** constructively owns in **Corp** as a result of Section 318(a).⁸

The Rule of Section 318(a)(1)—`num_family`

The procedure `num_family(Client, Corp, Number, zzzz)` determines the **Number** of shares that **Client** owns in **Corp** as a result of the family attribution rules set forth in Section 318(a)(1). First, the procedure checks to determine whether **Client** is an individual. If **Client** is not an individual, the procedure quickly determines that the number of shares of stock that **Client** owns as a result of the family attribution rules is zero. In this regard, if the user has told TA that **Client** is a partnership, estate, trust, grantor trust, or corporation, then the computer knows that the **Client** is not an individual.

If **Client** is an individual, `num_family` then asks the user for the names of **Client's** spouse, children, grandchildren, and parents. This is done by the procedure `family_of(Client, List)` where **List** is a list of all family members of **Client**.⁹

This procedure for determining the family members of **Client** is presently very crude. In particular, a more extensive program would contain the rule that a child includes a legally adopted child of an individual. Furthermore, a more sophisticated program would contain rules for determining the various family relationships between individuals. For example, it could contain rules such that if the program knew that John is the son of Dean then the computer could automatically infer that Dean is the parent of John. These rules have not been included in the present program due to memory constraints.

Once TA has determined the family members of **Client**, it then calls the procedure `family_own(Client, Corp, List, M, N)`. This procedure recursively goes down the **List** of family members and determines the number of shares of stock of **Corp** that each family member actually or constructively, as a

result of Section 318(a), owns. In determining the number of shares that a family member constructively owns, the program uses only the rules of Section 318(a)(2) and 318(a)(4). In other words, the computer knows that pursuant to Section 318(a)(5)(B), stock constructively owned by an individual as a result of the family attribution rules may not be considered as actually owned by the individual for purposes of again applying the family attribution rules. Furthermore, TA knows that the attribution rules of Section 318(a)(3) cannot apply, because those rules attribute stock ownership to an entity such as an estate, trust, or corporation. As a result, `family_own` simply calls `num_from` (the Section 318(a)(2) rule) and `num_options` (the Section 318(a)(4) rule).

Once TA determines the number of shares that each family member actually and constructively owns in **Corp**, `num_family` returns the total number of such shares.

The Rule of Section 318(a)(2)—`num_from`

The Prolog procedure `num_from(Client, Corp, Number, OldClient)` determines the **Number** of shares that **Client** owns of **Corp** as a result of the "from" attribution rules set forth in Section 318(a)(2). The procedure first calls `entity_of`. `entity_of` determines the names of all partnerships, estates, trusts, grantor trusts, and corporations in which **Client** has an interest and which may actually or constructively own stock of **Corp**. For example, assume that John is the **Client** and Corporation X is the **Corp**. Then TA will ask questions such as:

Is John a 50 percent or more shareholder in a corporation that owns, actually or constructively, stock in Corporation X?

If the user answers yes (by typing y) to such a question, the computer will then ask the user for the name of the entity and automatically add the type of the entity to the computer's data base. For example, if the user states that Corporation Y may actually or constructively own stock in Corporation X in response to the above question, the computer will add the fact `corporation(Corporation_Y)` to its data base. Finally, `entity_of` asks the user what percentage interest **Client** has in the entity in question. This is necessary because the rules set forth in Section 318(a)(2) state that stock owned by an entity is considered to be proportionately owned by its beneficiaries.

It should be noted that TA currently does not automatically determine whether a client owns a 50% or more ownership interest in a given corporation. It simply asks the user whether there is such a corporation. In particular, the 50% ownership requirement must be determined by applying the Section 318(a) attribution rules. This rule presently is not in the system.

After TA determines the identity and percentage ownership interest of **Client** in entities that may actually or constructively own stock in **Corp**, it then calls the procedure `entity_own(Client, Corp, List, M, N, OldClient)`, which determines the number of shares that each such entity in **List** actually or constructively owns in **Corp**. Once this is done, `entity_own` determines **Client's** proportional interest in such ownership. In other words, `entity_own` first calls `nu-`

Write it once!

MasterFORTH

Portable programming environment



Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will

protect your investment. If you write for others, it will expand your marketplace.



MasterFORTH is a state-of-the-art implementation of the Forth computer language.

Forth is interactive - you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code



let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

CP/M

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.

MasterFORTH standard package

Macintosh	\$125
IBM PC and PC Jr. (MS DOS 2.1)	125
Apple II, II+, IIe, IIc (DOS 3.3)	100
CP/M 2.X (in several formats)	100
Commodore 64	100

Extensions

Floating Point (1984 FVG standard)	\$40
Graphics (Apple II series)	40
Module relocater (with utility sources)	60
Printed source listing (each)	35

Publications

<i>Mastering Forth</i> (additional copies)	\$18
<i>Thinking Forth</i> by Leo Brodie	16
<i>Forth-83 International Standard</i>	15
<i>Rochester Bibliography</i> , 2nd ed.	15
<i>1984 Rochester Conference</i>	25
<i>1984 J1 of Forth Appl. & Res.</i> 2(2)	15
<i>1983 FORML Conference</i>	25



MICROMOTION

12077 Wilshire Blvd., #506
Los Angeles, CA 90025
(213) 821-4340

ASSEMBLE 3-6 TIMES FASTER ON THE IBM PC

Introducing **FAST ASSEM-86™**, the first Editor/Assembler for the IBM PC and PC compatibles. **FAST ASSEM-86™ (FASM)** is significantly faster and easier to use than the IBM Macro-Assembler (MASM). Whether you are new to assembly language and want to quickly write a small assembly language routine, or are an experienced MASM user tired of waiting months to assemble large files, **FAST ASSEM-86** will bring the excitement back to assembly language.

FAST ASSEM-86 IS MUCH FASTER:

- How fast is **FASM™**? The graph below shows relative assembly times for a 48K source file. For large files like this we blow MASM's doors off at 3 times their speed. For smaller 8K files we positively vaporize them at 6 times their speed.

FASM™ (110 sec.)
MASM v1.0 (340 sec.)

- **FAST ASSEM-86** is faster for the following reasons: (1) Written entirely in assembly language (unlike MASM). (2) Editor, assembler and source file always in memory so you can go instantly from editing to assembling and back. (3) Eliminates the time needed to LINK programs. Executable .COM files can be created directly. (Also creates .OBJ files completely compatible with the IBM linker).

FAST ASSEM-86 IS EASIER TO USE:

FASM includes many other features to make your programming simpler.

- Listings are sent directly to screen or printer. Assemblies can be single stepped and examined without having to leave the editor.
- Access the built in cross reference utility from the editor.
- Full support of 186 and 286 (real mode) instructions.
- Both Microsoft and 8087 floating point formats are supported. 8087 and 287 instructions supported directly without macros for faster assembly.
- Calculator mode: Do math in any radix even using symbols from the symbol table.
- Direct to memory assembly feature lets you test execute your code from editor.
- Coming soon: A coordinated symbolic debugger.

COMPATIBILITY: **FASM** is designed for source code compatibility with MASM and supports most of its important features.

Introductory Price \$ 99

Dealer inquires welcome

Speedware™

916-966-6247
Box D2, 2931 Northrop Avenue
Sacramento, CA 95825

IBM, Microsoft trademarks of IBM Corp., Microsoft Corp. respectively.

Circle no. 97 on reader service card.



Starlight FORTH

• Apple

• Atari

• AIM 65

• Commodore

• Ohio Scientific

for
65SC802 & 65SC816

The new 16 bit CMOS 6502's from
Western Design Center and GTE

- **Highly Extended fig-FORTH**
- **Full Double Number Set - Strings**
- **Optional Tools: Meta-Cross-Compiler with Assembler and Debugger**
- **Multi-Tasking Super Fast FORTH-83 soon**

Your present 8 bit system will run faster

with 16 bit software (FORTH) and a 65SC802

and still runs your existing 8 bit software

Starlight FORTH Systems

15247 N. 35th St.

Phoenix, AZ 85032

(602) 992-5181

Circle no. 110 on reader service card.

Circle no. 62 on reader service card.

num_a_own(**H**, **Corp**, **N1**) to determine the number of shares, **N1**, which the entity, **H**, actually owns in **Corp**. For example, if **Client** owns a 50% interest in a partnership, **PS**, and **PS** owns 100 shares of the stock in **Corp**, **entity_own** will determine that **Client** constructively owns 50 shares of stock in **Corp** as a result of Section 318(a)(2)(A).

For each entity in which **Client** has an interest and which may actually or constructively own stock of **Corp**, **entity_own** will then use the attribution rules of Section 318(a)(2) and 318(a)(4) to determine the number of shares of stock that the entity constructively owns. These rules account for the reattribution rules set forth in Section 318(a)(5)(A). **entity_own**, however, does not include the constructive ownership rule set forth in Section 318(a)(3) because Section 318(a)(5)(C) provides that stock constructively owned by an entity by reason of the application of the "to" attribution rules set forth in Section 318(a)(3) will not be considered as owned by the entity for purposes of applying the "from" attribution rules of Section 318(a)(2).

The problem of reattribution that is inherent in Section 318(a) has been easily handled in the Prolog rules because Prolog permits a recursive definition of procedures. Thus, the procedure **num_from** was able to call itself as applied to a different entity and, as a result, accounts for the reattribution rules of Section 318(a)(5)(A).

The Rule of Section 318(a)(3)—num_to

The procedure **num_to**(**Client**, **Corp**, **Number**) determines the **Number** of shares that **Client** constructively owns of **Corp** as a result of the "to" attribution rules set forth in Section 318(a)(3). Because these rules apply only if **Client** is not an individual, **num_to** returns zero if **Client** is an individual. Furthermore, if **Client** and **Corp** are the same, **num_to** again returns zero.

If **Client** is not an individual and is not the same as **Corp**, **num_to** then forms a list of all those beneficiaries of **Client** who may own stock, actually or constructively, in **Corp**. This procedure is similar to the procedure **entity_of** discussed above. The only difference is that if the procedure determines that the entity is a trust, it then asks the user whether the beneficiary's interest in the trust is a "remote contingent interest." If the user answers yes, the beneficiary is not included in the list of beneficiaries who may own stock in **Corp**.¹⁰

Once **TA** has determined the beneficiaries of **Client** who may actually, or constructively, own stock of **Corp**, **num_to** then determines the number of shares of stock that each such person actually and constructively, as a result of Sections 318(a)(1)–(4), owns of **Corp**. In other words, **num_to** calls a procedure **to_entity_own**, which then calls **num_a_own**, **num_family**, **num_from**, **num_to**, and **num_options**. This accounts for the reattribution rules set forth in Section 318(a)(5)(A).

The Rule of Section 318(a)(4)—num_options

Finally, the procedure **num_options**(**Client**, **Corp**, **Number**) handles the "options" attribution rules set forth in Section 318(a)(4). This procedure simply asks the user whether **Client** has an option, or an option to acquire an option, to acquire stock of **Corp**. If the user answers yes, **num_options**

then asks the user how many shares of **Corp** stock **Client** has an option on.

Examples

Listing Two contains several computer runs for various factual situations. These examples have been taken from the examples set forth in the Treasury Regulations under Section 318(a). The user input has been underlined for clarity.

Listing Three (page 84) contains an example of a run in which the procedures **num_a_own**, **num_con_own**, **num_family**, **num_from**, **num_to**, and **num_options** have been "traced." This permits the reader to see how Prolog recursively applies the various procedures in the case of a reattribution problem.

Simple Explanations

Although the Section 318(a) program correctly determines the number of shares of stock of a given corporation that a person constructively owns as a result of the Section 318(a) attribution rules, most attorneys using the program will also want to know **how** the computer reached its conclusion. For example, if a senior partner asks an associate to determine the number of shares of stock of a given corporation that an individual owns as a result of the Section 318(a) attribution rules and the associate simply gives the number, the senior partner would undoubtedly ask the associate to prepare a memo outlining the reasoning he or she used to arrive at the number of shares. Furthermore, if the Section 318(a) program were used to prepare a Tax Court petition, the attorney would need to present a well-reasoned opinion supporting his or her position. This could be done only by explaining how the Section 318(a) attribution rules applied to the facts in question. Finally, if the attorney simply relied upon the numeric output from the program and did not review the computer's reasoning, the attorney could leave himself or herself open to a malpractice suit. Thus, any useful "knowledge-based" system in the legal field must permit the user to ask the computer to explain **how** it reached its conclusion.

The issue of how to have the computer generate useful explanations is quite complex. For example, explanations by human experts are usually tailored to their audiences. In particular, an associate would use a different (and more technical) explanation of how he or she arrived at the number of shares of stock that an individual owns as a result of the Section 318(a) attribution rules if he or she were writing a memo to a senior tax partner as opposed to explaining the problem to a client. Furthermore, in most situations, the high level domain rules that are incorporated into the computer program may not adequately contain their justifications. Although these problems present a number of interesting questions, which should be addressed by future research, they will not be considered in this paper.¹¹

As will be illustrated below, most of the above complexities that arise in generating useful explanations have been avoided in **TA** because the rules set forth in Section 318(a) are justifications for themselves. In other words, the tax rules set forth in the Internal Revenue Code generally need no further justification other than the fact that Congress enacted such statutory provisions. Furthermore, the complex

problem of dealing with the interpretation of case law is avoided because there are very few cases under Section 318(a).

The remainder of this part of the paper will discuss how the Prolog program in Listing Four (page 87) implements a simple explanation facility.¹²

Answering How the Computer Reached Its Conclusion

In order to understand how TA explains **how** it arrives at a given conclusion, consider the following fact situation. Assume that TA is asked how many shares of stock of Corporation X the client, Dean, owns as a result of the family attribution rules. Furthermore, assume that Dean has only one family member, his spouse, Mary. Finally, assume that Mary actually owns 1,000 shares of Corporation X stock. Based on these facts, TA will explain its reasoning as follows:

START OF EXPLANATION

The number of shares of Corporation X stock constructively owned by Dean as a result of Section 318(a)(1) is 1,000, determined as follows:

An individual is deemed to own the stock owned by his family. Section 318(a)(1).

FACT: Dean is an individual.

A spouse is a family member. Section 318(a)(1)(A)(i).

FACT: Mary is the spouse of Dean.

FACT: Mary actually owns 1,000 shares.

FACT: Dean has no more family members.

END OF EXPLANATION

As can be seen, the explanation generated by TA applies the rules set forth in Section 318(a)(1) to the facts in question.

In order to explain its reasoning, each rule of TA contains an extra argument, **H**, which contains a list of descriptions of the Prolog rules that were successful when the program finally reached its conclusion.¹³ For example, when **num_a_own** determines how many shares of Corporation X stock Mary actually owns, it adds the description:

FACT: Mary actually owns 1,000 shares.

to the list of descriptions.

Furthermore, whenever a Prolog rule is successful, the rule adds a description of itself as the first element in the list for generating the explanation. For example, the predicate **spouse_own** adds the description found by **get_rule(2,Z)** to the head of the list once it has determined both that Dean has a spouse and the number of shares of stock the spouse actually owns. In particular, **get_rule(2,Z)** simply sets the first element of the list to the description:

VANCE info systems is pleased to announce
THE MOST COMPLETE C FUNCTION LIBRARY AVAILABLE TO DATE!

C FUNCTION LIBRARY

"C lib" is the most functional library available for software written in C, providing over 200 routines, extending the capabilities of C on the IBM PC. The library is available under the DeSmet (C Ware) C88 compiler, will soon be available in MicroSoft C, Lattice C and other C compilers, and runs using MS Dos 1.1 and later versions.

Featuring:

- Unix/Xenix Compatibility
- Windowing Library
- Math Functions
- String Functions
- Asynchronous Buffered, Interrupt-driven Communications
- 8087/MicroSoft Floating Point conversions
- And much more

Documentation is offered in an easy to use printed manual or on disk for personal printing needs, complete with programming examples and follow-up demo programs.

The "C lib" C FUNCTION LIBRARY is offered at only \$145, less than most available today.

For further information on "C lib" please call or write us.

VANCE info systems

2818 clay street • san francisco, california 94115 • (415)922-6539

IBM is a trademark of International Business Machines Corp. C88 is a trademark of Computer Innovations, Inc. Lattice is a trademark of Lattice Inc. Xenix, MicroSoft C and MS DOS are trademarks of MicroSoft Inc. Unix is a trademark of Bell Labs, Inc. "C lib" is a trademark of VANCE info systems.

PROLOG V

INTRODUCTORY OFFER

Valid through
May 31, 1985

ONLY
\$69⁹⁵

Examine the
documentation for
30 days and return
with disk still sealed
for full refund, if not
fully satisfied.

PHONE ORDERS:
(619) 483-8513



CHALCEDONY
SOFTWARE

5580 LA JOLLA BLVD.
SUITE 126 A
LA JOLLA, CA
92037

Prolog
Artificial Intelligence
Programming Language

Full Prolog as defined in Clocksin
& Mellish, *Programming in
Prolog*, Springer-Verlag, Berlin
Heidelberg New York, 1981.

Complete documentation with primer
and working-program examples.

Interpreter for IBM PCs® and compatibles.

IBM PC® is a registered trademark of IBM Corporation.

<input type="checkbox"/> PAYMENT ENCLOSED \$ _____
CA residents add 6% sales tax
<input type="checkbox"/> CHARGE MY: <input type="checkbox"/> MasterCard <input type="checkbox"/> Visa
Card No. _____ Exp. Date _____
Signature _____
Mr./Mrs./Ms. _____ (please print full name)
Address _____
City/State/Zip _____ 842DD

A spouse is a family member. Section 318(a)(1)(A)(i).

Once TA has determined how many shares of stock Dean constructively owns as a result of the Section 318(a)(1) family attribution rules, it asks the user if he or she wants an explanation of its conclusion. If the user answers yes, TA calls the predicate **how**, which simply prints each element of the list **H** containing the descriptions. (See Listing Four).

Answering Why a Given Question Is Asked

In the full Section 318(a) program in Listing One, TA asks the user for only those facts required to reach its conclusion. To make the program more useful, the user should be able to ask the computer **why** it asks a given question. For example, when the computer asks the user:

Is Dean an individual?

the user should be able to ask why the computer needs to know this information.

In the prototype program in Listing Four, the computer will answer **why** by returning:

An individual is deemed to own the stock owned by his family. Section 318(a)(1).

In other words, TA is telling the user that it needs to know if Dean is an individual in order to determine whether it should apply the family attribution rules of Section 318(a)(1). If the user is still not satisfied with this explanation, the user can again type **why** in response to the question: **Is Dean an individual?** This time, the program will return:

This is what the Internal Revenue Code provides!

In other words, in the simple Section 318(a) program, the ultimate explanation of any rule is that the Internal Revenue Code contains the rule.

In order to implement the **why** feature, an additional argument, **W**, is added to each rule. This argument contains a list of all rules that have been used by the program up to the point at which the computer asks for further information. In the above example, the program has just entered the rule **num_family**.

TA first attempts to satisfy this rule by determining whether **Client** is an individual. As a result, it asks the question:

Is Dean an individual?

At this point in the program, the argument **W** contains a list with one element, the description:

An individual is deemed to own the stock owned by his family. Section 318(a)(1).

Thus, when the user types **why** in response to the question, Prolog simply returns the first element of the list, **W**, and then asks the question again. If the user again asks **why**, the

program will print the next element of the list and again ask the question. When the list contains no further elements, the program will simply explain that:

This is what the Internal Revenue Code provides!

See Listing Five (page 90) for further examples of explanations.

Providing Help When the Computer Asks Questions

If the attorney using the program is not familiar with the given area of tax law, he or she may need **help** in answering some questions that the computer asks. For example, in order to keep the questions short, TA simply asks the question:

Does Dean have a spouse?

The user may need some assistance in determining how the word **spouse** is defined. The present version of TA permits the user to type **help** in response to the computer's request for information. For example, if the user types **help** in response to the question:

Does Dean have a spouse?

the computer will answer:

A spouse does not include one who is legally separated from Dean under a decree of divorce or separate maintenance. Section 318(a)(1)(A)(i).

Thus, TA gives the user some assistance in determining how the word **spouse** is defined for purposes of Section 318(a). The present system, however, does not further define the terms "legally separated," "decree of divorce," or "separate maintenance." For example, TA would not assist the user in determining whether an interlocutory decree of divorce under California law constitutes a "decree of divorce" for purposes of Section 318(a). It appears, however, that it would be possible to build further **help** facilities into the program if additional memory were available.

Finally, it should be recognized that this **help** facility may be insufficient in an actual system designed to assist attorneys because it requires the attorney to know when to ask for help. In particular, the attorney might simply assume that the word **spouse** for purposes of Section 318(a) includes any person who is married to the client even if legally separated from him or her. Thus, unless the user knew to ask for help, he or she might incorrectly answer the question.

Conclusion

This paper has discussed two Prolog programs that implement the rules contained in Section 318(a). Unlike other computer programs that have been developed in various legal fields, the major purpose of developing TA has been to illustrate that Prolog may be used to implement legal reasoning systems in a fairly straightforward manner. Furthermore, this paper has attempted to show that Prolog may also be used to implement simple explanation facilities for knowl-

NEW!

RELOCATABLE Z-80 MACRO ASSEMBLER FROM MITEK

It's a real bargain! Here's why:

- Only \$49.95 plus shipping
- 8080 to Z-80 Source Code Converter
- Generates Microsoft compatible REL files or INTEL compatible hex files
- Compatible with Digital Research macro assemblers MAC & RMAC
- Generates Digital Research compatible SYM files
- Full Zilog mnemonics
- INCLUDE and MACLIB files
- Conditional assembly
- Separate data, program, common and absolute program spaces
- Customize the Macro Assembler to your requirements with installation program
- Cross-reference Generation
- Z-80 Linker and Library Manager for Microsoft compatible REL files available as a total package with Macro Assembler for only \$95.00 plus shipping
- Manual only is \$15

TO ORDER, CALL TOLL FREE: 1-800-367-5134, ext. 804
For information or technical assistance: 1-808-623-6361

Specify desired 5 1/4" or 8" soft-sectored format. Personal check, cashier's check, money order, VISA, MC, or COD welcomed. Include \$5 for postage and handling.

MITEK P.O. Box 2151
Honolulu, HI 96805

Z-80 is a trademark of Zilog, Inc. MAC, RMAC, and ZSID are trademarks of Digital Research, Inc.

Circle no. 66 on reader service card.

BACKUP PROTECTED SOFTWARE WITH COPY II MAC™

From the team who gave you COPY II PLUS and COPY II PC comes a powerful, easy-to-use disk backup utility for your MACINTOSH computer. Features include:

- Bit Copy Program
- Make Files Visible/Invisible
- Sector/File Editor
- Format/Verify Disks
- Repair Damaged Disks
- Copy Protect/Unprotect
- Undelete Files
- Copy Files/Disk

See us at
WCCF in
Booth #1045

Increase the power of your
MACINTOSH... use **COPY II MAC**

Available at your local dealer or direct from us.

**CENTRAL POINT
Software, Inc.**

ONLY
\$39.95

PLUS \$3.00 SHIPPING, HANDLING

9700 S.W. Capitol Highway, #100/Portland, OR 97219

(503) 244-5782
M-F 8-5:30 (W. Coast Time)



CHECK, COD WELCOME
(Prepayment Required)

This product is provided for the purpose of enabling you to make archival copies only.

Circle no. 6 on reader service card.

LISP FOR THE IBM PERSONAL COMPUTER.

THE PREMIER LANGUAGE
OF ARTIFICIAL
INTELLIGENCE FOR
YOUR IBM PC.

■ DATA TYPES

Lists and Symbols
Unlimited Precision Integers
Floating Point Numbers
Character Strings
Multidimensional Arrays
Files
Machine Language Code

■ MEMORY MANAGEMENT

Full Memory Space Supported
Dynamic Allocation
Compacting Garbage Collector

■ FUNCTION TYPES

EXPR/FEXPR/MACRO
Machine Language Primitives
Over 190 Primitive Functions

■ IO SUPPORT

Multiple Display Windows
Cursor Control
All Function Keys Supported
Read and Splice Macros
Disk Files

■ POWERFUL ERROR RECOVERY

■ 8087 SUPPORT

■ COLOR GRAPHICS

■ LISP LIBRARY

Structured Programming Macros
Editor and Formatter
Package Support
Debugging Functions
.OBJ File Loader

■ RUNS UNDER PC-DOS 1.1 or 2.0

IQLISP

5 1/4" Diskette
and Manual _____ \$175.00
Manual Only _____ \$ 30.00

Integral Quality

P.O. Box 31970
Seattle, Washington 98103-0070
(206) 527-2918

Washington State residents add sales tax.
VISA and MASTERCARD accepted.
Shipping included for prepaid orders.

edge-based programs.

TA achieves its degree of success, however, mostly because the tax rules of Section 318(a) are very well defined and not subject to interpretation by a large number of cases. It is presently unclear how the rules set forth in case law could be incorporated into the present framework in an area where case law is at least as important as the statutory provisions.

The prototype programs also illustrate that it is possible to design interfaces to legal reasoning programs that permit a user who is not a computer expert to use the program. In particular, more effort has gone into devising a suitable user interface than into the more complex issues of representing the rules and knowledge required to implement the Section 318(a) programs. In narrow, well-defined areas of the law, however, this method may be acceptable. If it is, programs that are useful to attorneys may be able to be developed much more quickly than currently appears possible.

Finally, it should be recognized that the explanation facilities described in this paper are extremely simple. Because the best explanation of Section 318(a) is in terms of the rules of that Section, however, the explanation facility of TA is probably adequate. In other situations, however, such simple explanations would clearly be inadequate.

Notes

¹ Unless otherwise stated, all Section references are to the Internal Revenue Code of 1954, as amended.

² See for example: L. T. McCarty, "Reflections on TAX-MAN: An Experiment in Artificial Intelligence and Legal Reasoning," *Harvard Law Review* (1977) 90, 837-93; A. Hustler, *Programming Law in Logic*, Department of Computer Science, University of Waterloo, Research Report CS-82-13 (1982).

³ CP/M-86 is a trademark of Digital Research, Inc.

⁴ Prolog-86 is a trademark of MICRO-AI.

⁵ It is unclear, however, whether the shares of stock owned by a grantor trust will also be attributed to the beneficiaries of the trust pursuant to Section 318(a)(2)(B)(i) discussed above. TA implicitly contains the rule that shares of stock owned by a grantor trust *are* attributed to the beneficiaries pursuant to Section 318(a)(2)(B)(i).

⁶ The programs discussed in this paper are written in Prolog-86. Prolog-86 is based on the University of New South Wales Prolog, which runs under Unix (Unix is a trademark of Bell Telephone Laboratories) and is similar in many ways to the original DECSys-10 Prolog written by L. M. Pereira, F. C. N. Pereira, and D. H. D. Warren. Prolog-86 currently runs under CP/M-86, and MSDOS and PC DOS. Prolog-86 is available through Solution Systems, 335-D Washington Street, Norwell, MA 02061. Those readers who are interested in learning more about Prolog should read the book *Programming In Prolog* by W. F. Clocksin and C. S. Mellish and published by Springer-Verlag.

⁷ As a result of restrictions on numbers in Prolog-86, the implementation language of TA, fractional shares are not allowed. Furthermore, the number of shares must be less than approximately 30,000. These restrictions could be lifted if the Prolog interpreter permitted real numbers (i.e.,

fractions) and "long" integers.

⁸ In Prolog, the operator **is** causes arithmetic evaluation. Thus, **N is N1 + N2 + N3 + N4** sets **N** to the sum of **N1**, **N2**, **N3**, and **N4** if **N** is previously uninstantiated.

⁹ A list is an ordered sequence of elements of any length. In Prolog, a list is represented as: **[a, b, c]**. **a**, **b**, and **c** are the elements of the list. The null list is represented as **[]**.

¹⁰ Section 318(a)(3)(B)(i). In this regard, TA does not assist the user in determining whether a beneficiary's interest is a "remote contingent interest" as opposed to a "vested" interest or a contingent interest that is not remote. In a larger system, such rules must be included because it is unlikely that a user of the system would know how Section 318(a)(3)(B)(i) defines a remote contingent interest.

¹¹ See for example: J. A. Goguen, J. L. Weiner, and C. Linde, "Reasoning and Natural Explanation," *Int. J. Man-Machine Studies* (1983) 19, 521-559; William R. Swartout, "XPLAIN: A System for Creating and Explaining Expert Consulting Programs," *Artificial Intelligence* (1983) 21, 285-325; and William J. Clancey, "The Epistemology of a Rule-Based Expert System—a Framework for Explanation," *Artificial Intelligence* (1983) 20, 215-251.

¹² Due to memory constraints, the Prolog program described in Listing Four only implements the family attribution rules of Section 318(a)(1). If the Prolog interpreter were able to access more than 128K of RAM, it would be fairly easy to include the simple explanation system in the full Section 318(a) program listed in Listing One.

¹³ The design of the explanation facilities set forth in this paper is based on the article by K. L. Clark and F. G. McCabe with an appendix by P. Hammond, "PROLOG: A Language for Implementing Expert Systems," *Machine Intelligence* (1982) 10, 455-475.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 194.

Listing One

```
% THE COMPLETE SECTION 318(a) PROLOG PROGRAM
% By Dean A. Schlobohm
% © Copyright 1984 by Dean A. Schlobohm. All Rights Reserved.
% Version 1.0
% Date: 6/11/84
% Bugs:
% 1. Does not include the rule that stock cannot count more
% than once.
% 2. When it asks for partners, etc. that own stock, you must
% enter all partners who actually or constructively may own
% stock.
% 3. Does not handle "How", "Help" and "Why"
% To Use: type "prolog sec318.pro<RET>" then after Prolog is
% loaded type "section318!<RET>"
```

```
% preliminary functions
```

```
append([],L,L) :- !.
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

```
appendall([],[]).
appendall([H],H).
appendall([H1,H2],L) :- append(H1,H2,L).
appendall([H1,H2|T],L) :-
    append(H1,H2,X),
    appendall(T,Z,! ,
    append(X,Z,L).
```

```
member(X,[X|_]) :- !.
member(X,[_|Y]) :- member(X,Y).
```

```
add(X) :- X,! .
add(X) :- asserta(X),! .
```

```
% The Interface Program.
```

```
section318 :-
    prompt(Old,' '),
    nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,
    print(' THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES
    OF'),
    print('STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A
    RESULT'),
    print('OF SECTION 318(a) OF THE INTERNAL REVENUE CODE. '),
    nl,nl,
    ask(' Enter the name of the Client. (One word) ',
    ,Client),nl,
    type(Client),
    ask(' Enter the name of the Corporation. (One word) ',
    ,Corp),nl,
    add(corporation(Corp)),!,
    num_a_own(Client,Corp,N_actual),
    num_con_own(Client,Corp,N_con),!,nl,nl,
    print(' The number of shares of ',Corp,' stock actually
    owned by'),
    print(Client,' is ',N_actual,'. '),nl,nl,
    print(' The number of shares of ',Corp,' stock
    constructively owned'),
    print('by ',Client,' is ',N_con,'. '),nl,nl,!,
    ask(' Do you want to try another Client? ',X),!,
    ((X = y) ; (X = 'Y')),!,nl,nl,
    prompt(_ ,Old),
    section318.
```

```
individual(Client) :- ind(Client).
individual(Client) :-
    (partnership(Client) ;
    estate(Client) ;
    trust(Client) ;
    grantor_trust(Client) ;
    corporation(Client)),
    !,fail.
```

(Continued on next page)

Tax Advisor (Listing Continued, text begins on page 64)

Listing One

```

individual(Client) :-
    type(Client),!,
    individual(Client).

type(Client) :-
    (ind(Client) ;
    partnership(Client);
    estate(Client) ;
    trust(Client) ;
    grantor_trust(Client) ;
    corporation(Client)),!.

type(Client) :-
    print('    Is ',Client,' an Individual, Partnership, Estate,
    Trust, '),
    prin('Grantor trust, or Corporation? (enter i, p, e, t, g,
    or c) '),
    ratom(X),nl,
    assert_type(X,Client),!.

assert_type(i,C) :- add(ind(C)),!.
assert_type(p,C) :- add(partnership(C)),!.
assert_type(e,C) :- add(estate(C)),!.
assert_type(t,C) :- add(trust(C)),!.
assert_type(g,C) :- add(grantor_trust(C)),!.
assert_type(c,C) :- add(corporation(C)),!.

% the rule for actual ownership

num_a_own(Client,Client,0) :- !.
num_a_own(Client,Corp,N) :-
    print('    How many shares of ',Corp,' stock does ',Client),
    prin('actually own? '),
    ratom(X),nl,
    integer(X),
    N = X,!.
num_a_own(Client,Corp,N) :-
    print('    Error -- You must enter an integer!'),nl,!,
    num_a_own(Client,Corp,N).

% the main rule of Section 318(a)

num_con_own(Client,Corp,Number) :-
    num_family(Client,Corp,N1),          % Section 318(a)(1)
    num_from(Client,Corp,N2,zzzz),      % Section 318(a)(2)
    num_to(Client,Corp,N3),             % Section 318(a)(3)
    num_options(Client,Corp,N4),!,      % Section 318(a)(4)
    Number is N1 + N2 + N3 + N4.

% the rule of Section 318(a)(1)

num_family(Client,Corp,Number) :-
    individual(Client),!,
    family_of(Client,List_of_family),
    family_own(Client,Corp,List_of_family,0,Number).
num_family(Client,Corp,0).

% definition of family_of

family_of(C,L) :-
    nl,
    prin('    Does ',C,' have a spouse, child, grandchild, or
    parent? '),
    ratom(X),nl,
    ((X = y) ; (X = 'Y')),!,
    ask('    What is the name of the first family member? '
    ,Y),nl,!,
    add(ind(Y)),
    family_of2(C,L2),!,
    append([Y],L2,L).
family_of(C,[]).

family_of2(C,L) :-
    nl,
    prin('    Does ',C,' have any more family members? '),
    ratom(X),nl,

```



```

((X = Y) ; (X = 'Y')),!,
ask('      What is the name of the next family member?
,Y),nl,!,
add(ind(Y)),
family_of2(C,L2),!,
append([Y],L2,L).
family_of2(C,[ ]).

% definition of family_own
% note Section 318(a)(5)(B) is included since family_own does
% not include num_family.

family_own(Client,Corp,[ ],N,N):- !.
family_own(Client,Corp,[P|T],M,N) :-
P /= Client,!,
num_a_own(P,Corp,N1),
num_fom(P,Corp,N2,zzzz),
option_lieu_family(Client,Corp,P,N3), % Section 318(a)(5)(D)
N4 is N1 + N2 - N3,
num_options(P,Corp,N5),
N6 is M + N4 + N5,
!,
family_own(Client,Corp,T,N6,N),!.
family_own(Client,Corp,[P|T],M,N) :-
family_own(Client,T,M,N).

option_lieu_family(Client,Corp,P,N) :-
print('      Does ',Client,' own an option (or an option to
acquire'),
print('an option) to acquire the stock in ',Corp,' which
',P),
prin('actually owns? '),
ratom(X),nl,
((X = Y) ; (X = 'Y')),!,
print('      How many shares of ',Corp,' stock does ',Client),
prin('have an option on? '),
ratom(N),nl,nl,!.

```

(Continued on next page)

QUALITY SOFTWARE AT REASONABLE PRICES

CP/M Software by
Poor Person Software

Keyed Sequential Files **\$39.95**

CP/M 2.2 Extension. Available to all languages with a BDOS interface. Variable length Records to 508 bytes, single key to 32 bytes, files to 256K bytes. Source for "C" language support library.

Poor Person's Menus **\$29.95**

Interactive menu design and edit program with "C" language source for runtime support library. Package includes "C" source for an interactive Mailing List Manager using Poor Person's Menus and separately purchased Keyed Sequential Files (compatible with Mailing Label Printer). Write your own interactive data manager programs.

Poor Person's Spooler **\$49.95**

All the function of a hardware print buffer at a fraction of the cost. Keyboard control. Spools and prints simultaneously.

Poor Person's Spelling Checker **\$29.95**

Simple and fast! 33,000 word dictionary. Checks any CP/M text file.

Mailing Label Printer **\$29.95**

Select and print labels in many formats.

Window System **\$29.95**

Application control of independent virtual screens.

All products require 48k CP/M 2.2 and are available on 8" and 5" Northstar formats, other 5" formats add \$5 handling charge. California residents include sales tax.

Poor Person Software

3721 Starr King Circle
Palo Alto, CA 94306
tel 415-493-3735

CP/M is a registered trademark of Digital Research

BIG DISCOUNTS FROM JOHN D. OWENS ASSOCIATES

MACROTECH MI-286: 80286/Z-80H DUAL PROCESSOR S-100 CPU BOARD: \$1,116

MACROTECH MEMORY MSR: 120NS, high-speed dynamic RAM with realistic pricing:

Works with CompuPro 8085/8088; MT-286 and others:
256K: \$556 512K: \$876

HOUSTON INSTRUMENTS: Plotters: DMP 41 OR 42: \$2,397; DMP 29: \$1,838

DIGITIZERS: DT11 \$694; DT114 \$750; DT11AA \$714

UCI S-100 RAM: ZENITH compatible 256K \$636; 512K \$916

PSCE S-100 68000 Single board computer: \$850

ILLUMINATED TECHNOLOGY S-100 COLOR GRAPHICS: \$1,116

RACTER: Interactive Conversational Software for IBM-PC. 4 page review in JAN 85 SCIENTIFIC AMERICAN called RACTER "... extremely funny" \$69.95 + \$3 UPS.

BIG DISCOUNTS on LOMAS, COMPUPRO, IMS, INTERCONTINENTAL MICRO, NEC APC, SEMIDISK, ADVANCED DIGITAL, ACKERMAN, SCION, and many others. Write or call for product literature and inventory sale list.

WE EXPORT: OVERSEAS CALLERS: TWX 710 588 2844
(OWENSASSOC NYK)

DOMESTIC AND OVERSEAS ORDERS TAKEN BY PHONE, LETTER, TELEX OR EASY LINK.
We accept VISA and Mastercard. Shipping \$5 per board in continental USA.

JOHN D. OWENS ASSOCIATES

12 SCHUBERT STREET STATEN ISLAND, NEW YORK 10305
(718) 448 6283 (718) 448 6298 (718) 448 2913
EASY LINK MAILBOX ADDRESS: 63840768


```

option_lieu_family(Client,Corp,P,0) :- !.

% the rule of section 318(a)(2)
% entity_of returns a list ((name percent) (name2 percent2)...)

num_from(Client,Corp,0,OldClient) :-
    nl,
    print('    Do you want to consider applying the rules of'),
    prin('Section 318(a)(2) to ',Client,'? (enter y or n) '),
    ratom(X),nl,
    ((X = n) ; (X = 'N')),!,
num_from(Client,Corp,N,OldClient) :-
    ps(P),
    entity_of(Client,Corp,L1,partnership,P),
    est(E),
    entity_of(Client,Corp,L2,estate,E),
    tr(T),
    entity_of(Client,Corp,L3,trust,T),
    gt(T1),
    entity_of(Client,Corp,L4,grantor_trust,T1),
    corp(C),
    entity_of(Client,Corp,L5,corporation,C),
    appendall([L1,L2,L3,L4,L5],L),
    entity_own(Client,Corp,L,0,N,OldClient).

entity_of(Client,Corp,L,Entity,Prompt) :-
    nl,
    print('    Is ',Client,Prompt),
    prin('which owns, actually or constructively, stock in ',
    ,Corp,'? '),
    ratom(X),nl,
    ((X = y) ; (X = 'Y')),!,
    prin('    What is the name of the ',Entity,'? '),
    ratom(Y),nl,
    add(Entity(Y)),
    prin('    What percentage interest does ',Client,' own? '),
    ratom(Z),nl,
    append([Y],[Z],Pl),!,
    entity_of(Client,Corp,L2,Entity,Prompt),!,
    append([Pl],L2,L).
entity_of(Client,Corp,[],Entity,Prompt).

ps(' a partner in a partnership').
est(' a beneficiary of an estate').
tr(' a beneficiary of a trust (not described in Section 401(a))').
gt(' the grantor of a grantor trust').
corp(' a 50 percent or more shareholder in a corporation').

% note Section 318(a)(5)(C) is included since entity_own does
% not include num_to

entity_own(Client,Corp,[],N,N,_) :- !.
entity_own(Client,Corp,[H,P|T],N,M,OldClient) :-
    H /= Client,
    H /= OldClient,
    num_a_own(H,Corp,N1),
    num_from(H,Corp,N2,Client),
    num_options(H,Corp,N3),
    N4 is (N1 + N2 + N3) * P / 100,    % this creates an error if
    % N1+N2+N3 > 600
    N5 is N4 + N,
    !,
    entity_own(Client,Corp,T,N5,M,OldClient).
entity_own(Client,Corp,[H|T],N,M,OldClient) :-
    entity_own(Client,Corp,T,N,M,OldClient).

% the rule of Section 318(a)(3)
% ben_of_entity returns a list (name name2 ...)

num_to(Client,Corp,0) :-
    not(individual(Client)),
    Client /= Corp,
    print('    Do you want to consider applying the rules of'),
    prin('Section 318(a)(3) to ',Client,'? (enter y or n) '),

```



```

    ratom(X),nl,
    ((X = n) ; (X = 'N'))),!.
num_to(Client,Corp,N) :-
    not(individual(Client)),
    Client /= Corp,!,
    ben_of_entity(Client,Corp,L1,partner,partnership),
    ben_of_entity(Client,Corp,L2,beneficiary,estate),
    ben_of_entity(Client,Corp,L3,beneficiary,trust),
    ben_of_entity(Client,Corp,L4,grantor,grantor_trust),
    ben_of_entity(Client,Corp,L5,shareholder,corporation),
    appendall([L1,L2,L3,L4,L5],L),
    to_entity_own(Client,Corp,L,0,N).
num_to(Client,Corp,0).

ben_of_entity(Client,Corp,L,Prompt,Entity) :-
    Entity(Client),
    print('      Is there a ',Prompt,' of ',Client,' who owns,
    actually'),
    prin('or constructively, stock in ',Corp,'? '),
    ratom(X),nl,
    ((X = y) ; (X = 'Y')),
    ((Entity = trust,
    ask('      Is the interest a remote contingent interest? '
    ,X2),nl,
    ((X2 /= y) , (X2 /= 'Y')))) ;
    (Entity = corporation,
    ask('      Does the shareholder own 50% or more of the stock?
    ,X3),
    nl,((X3 = y) ; (X3 = 'Y')))) ;
    (Entity /= trust , Entity /= corporation)),
    prin('      What is the name of the ',Prompt,'? '),
    ratom(Y),nl,nl,
    type(Y),!,
    ben_of_entity(Client,Corp,L2,Prompt,Entity),!,
    append([Y],L2,L).
ben_of_entity(Client,Corp,[],Prompt,Entity) :- !.

to_entity_own(Client,Corp,[],N,N) :- !.
to_entity_own(Client,Corp,[H|T],N,M) :-
    H /= Client,
    num_a_own(H,Corp,N1),
    num_family(H,Corp,N2),
    num_from(H,Corp,N3,Client),
    num_to(i,Corp,N4),
    num_options(H,Corp,N5),
    N6 is N + N1 + N2 + N3 + N4 + N5,
    !,
    to_entity_own(Client,Corp,T,N6,M).
to_entity_own(Client,Corp,[H|T],N,M) :-
    !,
    to_entity_own(Client,Corp,T,N,M).

% the rule of Section 318(a)(4)

num_options(Client,Corp,N) :-
    nl,
    print('      Does ',Client,' own an option (or an option to
    acquire'),
    prin('an option) to acquire the stock in ',Corp,'? '),
    ratom(X),nl,
    ((X = y) ; (X = 'Y')),!,
    print('      How many shares of ',Corp,' stock does ',Client),
    prin('have an option on? '),
    ratom(Y),nl,
    integer(Y),!,
    num_options(Client,Corp,M),
    N is M + Y,!.
num_options(Client,Corp,0).

```

End Listing One

Listing Two

```

: % EXAMPLE 1. Reg. Section 1.318-2(b).
:
: section318!

```

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a) OF THE INTERNAL REVENUE CODE.

(Continued on next page)

Tax Advisor (Listing Continued, text begins on page 64)
Listing Two

Enter the name of the Client. (One word) H

Is H an Individual, Partnership, Estate, Trust, Grantor trust, or Corporation? (enter i, p, e, t, g, or c) i

Enter the name of the Corporation. (One word) Corporation

How many shares of Corporation stock does H actually own? 25

Does H have a spouse, child, grandchild, or parent? y

What is the name of the first family member? W

Does H have any more family members? y

What is the name of the next family member? GS

Does H have any more family members? n

How many shares of Corporation stock does W actually own? 25

Do you want to consider applying the rules of Section 318(a)(2) to W? (enter y or n) n

Does H own an option (or an option to acquire an option) to acquire the stock in Corporation which W actually owns? n

Does W own an option (or an option to acquire an option) to acquire the stock in Corporation? n

How many shares of Corporation stock does S actually own? 25

Do you want to consider applying the rules of Section 318(a)(2) to S? (enter y or n) n

Does H own an option (or an option to acquire an option) to acquire the stock in Corporation which S actually owns? n

Does S own an option (or an option to acquire an option) to acquire the stock in Corporation? n

How many shares of Corporation stock does GS actually own? 25

Do you want to consider applying the rules of Section 318(a)(2) to GS? (enter y or n) n

Does H own an option (or an option to acquire an option) to acquire the stock in Corporation which GS actually owns? n

Does GS own an option (or an option to acquire an option) to acquire the stock in Corporation? n

Do you want to consider applying the rules of Section 318(a)(2) to H? (enter y or n) n

Does H own an option (or an option to acquire an option) to acquire the stock in Corporation? n

The number of shares of Corporation stock actually owned by H is 25.

The number of shares of Corporation stock constructively owned by H is 75.

Do you want to try another Client? n

: & EXAMPLE 2. Reg. Section 1.318-2(c), Example(2).
:

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a) OF THE INTERNAL REVENUE CODE.

Enter the name of the Client. (One word) A

(Continued on page 82)

Dr. Dobb's Journal

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP, MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify) _____

B. Which languages are you MOST interested in?

- ☐ BASIC ☐ C ☐ PL/I
- ☐ Fortran ☐ LISP ☐ APL
- ☐ COBOL ☐ Prolog ☐ Logo
- ☐ Pascal ☐ Ada ☐ Smalltalk
- ☐ Modula-2 ☐ Forth ☐ Other _____

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other _____

March 1985, #101

Dr. Dobb's Journal

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP, MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify) _____

B. Which languages are you MOST interested in?

- ☐ BASIC ☐ C ☐ PL/I
- ☐ Fortran ☐ LISP ☐ APL
- ☐ COBOL ☐ Prolog ☐ Logo
- ☐ Pascal ☐ Ada ☐ Smalltalk
- ☐ Modula-2 ☐ Forth ☐ Other _____

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other _____

Reader Service Card

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	<input type="checkbox"/>	<input type="checkbox"/>
Commodore	<input type="checkbox"/>	<input type="checkbox"/>
Digital Equipment/DEC	<input type="checkbox"/>	<input type="checkbox"/>
Heath/Zenith	<input type="checkbox"/>	<input type="checkbox"/>
Hewlett-Packard	<input type="checkbox"/>	<input type="checkbox"/>
IBM	<input type="checkbox"/>	<input type="checkbox"/>
Macintosh	<input type="checkbox"/>	<input type="checkbox"/>
Radio Shack/Tandy TRS 80	<input type="checkbox"/>	<input type="checkbox"/>
Texas Instruments	<input type="checkbox"/>	<input type="checkbox"/>
Other (Specify)	<input type="checkbox"/>	<input type="checkbox"/>
None	<input type="checkbox"/>	<input type="checkbox"/>

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

- ☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

- ☐ Yes ☐ No

H. Subscriber

- ☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	24	25	26	2	
28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	5
55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	8
82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	10
109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	13

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

P.O. BOX 13851

PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

Dr. Dobb's Journal

P.O. BOX 13851

PHILADELPHIA, PA 19101

"Ouvrez les fenêtres!"*

Introducing **MATIS™**, the powerful new developmental system from France.

A complete and meticulously detailed program to make a programmer's work easier, faster, and... but of course... better.

- ☐ Window Management Systems
- ☐ Screen Generator
- ☐ Expanded Basic Commands
- ☐ Can be accessed from other languages
- ☐ 100% Assembler
- ☐ Automatic Scrolling in Windows
- ☐ Virtual Page larger than screen (up to 65534 rows x 65534 columns)
- ☐ Save or Print Pages
- ☐ MS-DOS
- ☐ 170 Page Manual (In English Mon Ami!)
- ☐ Only \$150.

ORDER BY MAIL—WRITE OR CALL FOR COMPLETE DESCRIPTION
No license fee.

Softway, Inc.

500 Sutter Street • Suite 222—C • San Francisco, CA 94102
Tel: (415) 397-4666 Telex: 880857

*"Open the windows!"

Circle no. 92 on reader service card.

INSIGHT™

EXPERT SYSTEMS

"INSIGHT is essentially the equivalent or better than any other tool available for the personal computer."

Paul Harmon, author of Expert Systems, Artificial Intelligence in Business

Turn your PC into an expert.

Give it Insight, or give it Insight 2.

Both let you create knowledge base systems using any PC-compatible text editor.

Insight not only simplifies access to lots of information, it analyzes and offers solutions. For entry-level operators it's a perfect procedural training package to help build and implement knowledge base software.

Insight 2 is more than just an "expert." It's a knowledge base engineering tool with application capabilities. It can call up Pascal programs, read and write dBASE II® files, and its decision-making process can tie in directly to your existing databases. Run-only versions also can be developed and distributed.

Two unique packages from the same expert idea.

Insight™ (\$95) and Insight 2™ (\$485) run on the IBM® PC, DEC® Rainbow, and Victor® 9000.



**Level
Five
Research, Inc.**

4980 South A-1-A

Melbourne Beach, Florida 32951

(305) 729-9046

Circle no. 53 on reader service card.

"This is a beautifully documented, incredibly comprehensive set of C Function Libraries."

— Dr. Dobb's Journal



COMPLETE SOURCES

- **PACK 1: Building Blocks I** \$149
250 Functions: DOS, Printer, Video, Asynch
- **PACK 2: Database** \$399
100 Functions: B-Trees, Variable Records
- **PACK 3: Communications** \$149
135 Functions: Smart-modem™, Xon/Xoff, Modem-7, X-Modem
- **PACK 4: Building Blocks II** \$149
100 Functions: Dates, Text Windows, Pull-down Menus, Data Compression
- **PACK 5: Mathematics I** \$99
35 Functions: Log, Trig, Square Root
- **PACK 6: Utilities I** \$99
Archive, Diff, Replace, Scan, Wipe (Executable Files only)

Lattice™, Microsoft™, DeSmet™, C1-86™ Compilers on IBM PC/XT/AT™
Small and Large Memory Models.
Credit cards accepted
(\$7.00 handling/Mass. add 5%)

**SOFTWARE
HORIZONS
INC.**

165 Bedford Street
Burlington, Mass. 01803
(617) 273-4711

NOVUM ORGANUM

Circle no. 90 on reader service card.

Tax Advisor (Listing Continued, text begins on page 64)
Listing Two

Enter the name of the Corporation. (One word) Corporation

How many shares of Corporation stock does A actually own? 75

Does A have a spouse, child, grandchild, or parent. n

Do you want to consider applying the rules of Section 318(a)(2) to A? (enter y or n) y

Is A a partner in a partnership which owns, actually or constructively, stock in Corporation? n

Is A a beneficiary of an estate which owns, actually or constructively, stock in Corporation? n

Is A a beneficiary of a trust (not described in Section 401(a)) which owns, actually or constructively, stock in Corporation? y

What is the name of the trust? Trust

What percentage interest does A own? 4

Is A a beneficiary of a trust (not described in Section 401(a)) which owns, actually or constructively, stock in Corporation? n

Is A the grantor of a grantor trust which owns, actually or constructively, stock in Corporation? n

Is A a 50 percent or more shareholder in a corporation which owns, actually or constructively, stock in Corporation? n

How many share of Corporation stock does Trust actually own? 25

Do you want to consider applying the rules of Section 318(a)(2) to Trust? (enter y or n) n

Does Trust own an option (or an option to acquire an option) to acquire the stock in Corporation? n

Does A own an option (or an option to acquire an option) to acquire the stock in Corporation? n

The number of shares of Corporation stock actually owned by A is 75.

The number of shares of Corporation stock constructively owned by A is 1.

Do you want to try another Client? n

: % EXAMPLE 3. Reg. Section 1.318-2(c), Example(4).
:

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a) OF THE INTERNAL REVENUE CODE.

Enter the name of the Client. (One word) A

Enter the name of the Corporation (One word) Corporation O

How many shares of Corporation_O stock does A actually own? 50

Does A have a spouse, child, grandchild, or parent? n

Do you want to consider applying the rules of Section 318(a)(2) to A? (enter y or n) y

Is A a partner in a partnership which owns, actually or constructively, stock in Corporation_O? n

Is A a beneficiary of an estate which owns, actually or constructively, stock in Corporation_O? n

Is A a beneficiary of a trust (not described in Section 401(a)) which owns, actually or constructively, stock in Corporation_O? n

Is A the grantor of a grantor trust which owns, actually or constructively, stock in Corporation_O? n

Is A a 50 percent or more shareholder in a corporation which owns, actually or constructively, stock in Corporation_O? y

What is the name of the corporation? Corporation M

What percentage interest does A own? 70

Is A a 50 percent or more shareholder in a corporation which owns, actually or constructively, stock in Corporation_O? n

How many shares of Corporation_O stock does Corporation_M actually own? 50

Do you want to consider applying the rules of Section 318(a)(2) to Corporation_M? (enter y or n) n

Does Corporation_M own an option (or an option to acquire an option) to acquire the stock in Corporation_O? n

Does A own an option (or an option to acquire an option) to acquire the stock in Corporation_O? n

The number of shares of Corporation_O stock actually owned by A is 50.

The number of shares of Corporation_O stock constructively owned by A is 35.

Do you want to try another Client? y

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a) OF THE INTERNAL REVENUE CODE.

Enter the name of the Client. (One word) Corporation M

Enter the name of the Corporation. (One word) Corporation O

How many shares of Corporation_O stock does Corporation_M actually own? 50

Do you want to consider applying the rules of Section 318(a)(2) to Corporation_M? (enter y or n) n

Do you want to consider applying the rules of Section 318(a)(3) to Corporation_M? (enter y or n) y

Is there a shareholder of Corporation_M who owns, actually or constructively, stock in Corporation_O? y

Does the shareholder own 50% or more of the stock? y

What is the name of the shareholder? A

Is there a shareholder of Corporation_M who owns, actually or constructively, stock in Corporation_O? n

How many shares of Corporation_O stock does A actually own? 50

Does A have a spouse, child, grandchild, or parent? n

Do you want to consider applying the rules of Section 318(a)(2) to A? (enter y or n) n

Does A own an option (or an option to acquire an option) to acquire the stock in Corporation_O? n

(Continued on next page)

Listing Two

Does Corporation_M own an option (or an option to acquire an option) to acquire the stock in Corporation_O? n

The number of shares of Corporation_O stock actually owned by Corporation_M is 50.

The number of shares of Corporation_O stock constructively owned by Corporation_M is 50.

Do you want to try another Client? n

End Listing Two

Listing Three

```
: % An Example From Reg. Section 1.318-4(c)(1)
:
: trace[num a own,num con own,num family,num from]!
: trace[num to,num options]!
:
: section318!
```

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT ACTUALLY AND CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a) OF THE INTERNAL REVENUE CODE.

Enter the name of the Client. (One word) Corporation Y

Is Corporation_Y an Individual, Partnership, Estate, Trust, Grantor trust, or Corporation? (enter i, p, e, t, g, or c) c

Enter the name of the Corporation. (One word) Corporation Z

```
C|>num_a_own('Corporation_Y', 'Corporation_Z', _6)
How many shares of Corporation_Z stock does Corporation_Y
actually own? 0
```

```
E|<num_a_own('Corporation_Y', 'Corporation_Z', 0)
C|>num_con_own('Corporation_Y', 'Corporation_Z', _4)
C|>num_family('Corporation_Y', 'Corporation_Z', _26)
R|>num_family('Corporation_Y', 'Corporation_Z', 0)
E|<num_family('Corporation_Y', 'Corporation_Z', 0)
C|>num_from('Corporation_Y', 'Corporation_Z', 0, zzzz)
```

Do you want to consider applying the rules of Section 318(a)(2) to Corporation_Y? (enter y or n) n

```
E|<num_from('Corporation_Y', 'Corporation_Z', 0, zzzz)
C|>num_to('Corporation_Y', 'Corporation_Z', 0)
```

Do you want to consider applying the rules of Section 318(a)(3) to Corporation_Y? (enter y or n) y

```
R|>num_to('Corporation_Y', 'Corporation_Z', _28)
Is there a shareholder of Corporation_Y who owns, actually or
constructively, stock in Corporation_Z? y
```

Does the shareholder own 50% or more of the stock? y

What is the name of the shareholder? A

Is A an Individual, Partnership, Estate, Trust, Grantor trust, or corporation? (enter i, p, e, t, g, or c) i

Is there a shareholder of Corporation_Y who owns, actually or constructively, stock in Corporation_Z? n

```
C||>num_a_own('A', 'Corporation_Z', _136)
How many share of Corporation_Z stock does A actually own? 0
```

```
E||<num_a_own('A', 'Corporation_Z', 0)
C||>num_family('A', 'Corporation_Z', _137)
```


New!

Now You Can Add ARTIFICIAL INTELLIGENCE

To Your Programs Using a Powerful Combination



By Elliot Schneider & Jack Park

Here's Your Chance to Profit by being on
the Forefront, Write 5th Generation Software

Learn How To:

- Create Intelligent Programs
- Use Inference Engines
- Write Stand Alone License Free Programs

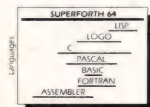
- Construct Rule Bases
- Do Knowledge Engineering
- Build Expert Systems

Write Intelligent Programs For:

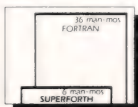
- Home Use
- Robotics
- Medical Diagnosis
- Education
- Intelligent CAI
- Scientific Analysis
- Data Acquisition
- Business
- Real Time Process Control
- Fast Games
- Graphics
- Financial Decisions

Extended Math Functions

- Fast ML Floating Point & Integer Math
- Double Precision 2E+38 with Auto. Sci Not.
- n*e* Logx Loge Sin Cos Tan SQRT/X...
- Matrix and Multidimensional Lattice Math
- Algebraic Expression Evaluator



Power of Languages Constructs
SuperForth 64 is more powerful than most other computer languages



Programming Time
SuperForth 64 Saves You Time and Money

Easy Graphics & Sound Words

- Hires Plotting
- Windows
- Split Screen
- Printer/Plotter Ctrl
- Sprite & Animation Editor
- Turtle Graphics
- Koala Pad Graphics Integrator
- Hires Circle, Line, Arc
- Music Editor
- Sound Control

Easy Control of all I/O...

- RS232 Functions
- Access all C-64 Peripherals

Utilities

- Interactive Interpreter
- Forth Virtual Memory
- Full Cursor Screen Editor
- Full String Handling
- Trace & Decompiler
- Conditional Macro Assembler
- Interrupt Routines
- Interactive Compiler
- Romable Code Generator
- 40K User Memory
- All Commodore File Types
- Conversational User Defined Commands

Great Documentation

- Easy to Read 350 pg. Manual with Tutorials
- Source Screen Provided
- Meets all MVP Forth-79 Industrial Standards
- Personal User Support



A Total Integrated Package

SMALL C FOR IBM-PC

Small-C Compiler Version 2.1 for PC-DOS/MS-DOS
Source Code included for Compiler & Library
New 8086 optimizations
Rich I/O & Standard Library

\$40

CBUG SOURCE LEVEL DEBUGGER FOR SMALL C

Break, Trace, and Change variables all on the source level
Source code included

\$40

Datalight

11557 8th Ave. N.E.
Seattle, Washington 98125
(206) 367-1803

ASM or MASM is required with compiler.
Include disk size (160K/320K), and DOS version with order.
VISA & MasterCard accepted. Include card no. & expiration date.
Washington state residents include 7.9% sales tax.
IBM-PC & PC-DOS are trademarks of International Business Machines
MS-DOS is a trademark of Microsoft Corporation.

SPECIAL INTRODUCTORY OFFER

only \$9900

203rd Value
Limited Time Offer

(*2nd Ship U.S.A. • *2nd C.O.D., U.S.A. • *15th Overseas)

Commodore 64 TM of Commodore • *Parsec Research

Circle no. 68 on reader service card.

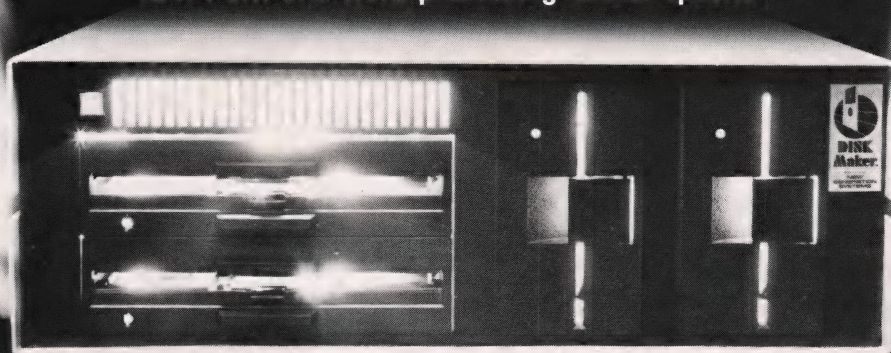
See Your Local
Computer Store
or CALL

(415) 651-3160
PARSEC RESEARCH
Box 1766 Fremont, CA 94538
Dealer Inquiries Invited

Circle no. 31 on reader service card.

The Most Affordable Disk Maker™ NEW! in the Universe

Now with over 25 MSDOS formats, 3 1/2" formats,
IBM PCAT and word processing format options



Disk Maker II shown
with opt. drives

Download fast, read over 200 formats easily, reformat rapidly

The more disk formats you work with, the more our Disk Maker™ system saves time and money by reading and/or writing disks in any of over 200 formats. No modems, no patches, no other special software necessary.

Disk Maker II is a complete, stand alone system with one 8" DSDD disk drive, one 48 tpi 5 1/4" DSDD disk drive, 6 MHz Z80B, 64K CP/M system with Disk Maker™ software. (96 tpi and second 8" drive optional.) Just plug in your terminal and make disks! Bundled software includes MicroShell™/MCALL-II communications software. Base price: \$3,395.

Supported with comprehensive, easy-to-read manual, software updates (\$50.00, all formats in revision), and additional drives and hard disk options.

Disk Maker™
prices from
\$1,695

Disk Maker I runs as a peripheral with an S-100 system and comes with S-100 controller board, one 48 tpi DSDD 5 1/4" disk drive, dual drive cabinet and power supply, cables and Disk Maker software. 96 tpi and 8" drives are optional. Base price: \$1,695.

**NEW
GENERATION
SYSTEMS**

1800 Michael Faraday Drive, Suite 206, Reston, VA 22090
(703) 471-5598 Order Line: (800) 368-3359 Dealer inquiries welcomed

Circle no. 76 on reader service card.

Listing Three

Does A have a spouse, child, grandchild, or parent? n

```
E|||<num_family('A', 'Corporation_Z', 0)
C|||>num_from('A', 'Corporation_Z', 0, 'Corporation_Y')
```

Do you want to consider applying the rules of Section 318(a)(2) to A? (enter y or n) y

```
R|||>num_from('A', 'Corporation_Z', _138, 'Corporation_Y')
```

Is A a partner in a partnership which owns, actually or constructively, stock in Corporation_Z? n

Is A a beneficiary of an estate which owns, actually or constructively, stock in Corporation_Z? n

Is A the grantor of a grantor trust which owns, actually or constructively, stock in Corporation_Z? n

Is A a 50 percent or more shareholder in a corporation which owns, actually or constructively, stock in Corporation_Z? y

What is the name of the corporation? Corporation X

What percentage interest does A own? 100

Is A a 50 percent or more shareholder in a corporation which owns, actually or constructively, stock in Corporation_Z? n

```
C|||>num_a_own('Corporation_X', 'Corporation_Z', _242)
How many shares of Corporation_Z stock does Corporation_X
actually own? 50
```

```
E|||<num_a_own('Corporation_X', 'Corporation_Z', 50)
C|||>num_from('Corporation_X', 'Corporation_Z', 0, 'A')
```

Do you want to consider applying the rules of Section 318(a)(2) to Corporation_X? (enter y or n) n

```
E|||<num_from('Corporation_X', 'Corporation_Z', 0, 'A')
C|||>num_options('Corporation_X', 'Corporation_Z', _244)
```

Does Corporation_X own an option (or an option to acquire an option) to acquire the stock in Corporation_Z? n

```
R|||>num_options('Corporation_X', 'Corporation_Z', 0)
E|||<num_options('Corporation_X', 'Corporation_Z', 0)
E|||<num_from('A', 'Corporation_Z', 50, 'Corporation_Y')
C|||>num_to('A', 'Corporation_Z', 0)
R|||>num_to('A', 'Corporation_Z', _139)
R|||>num_to('A', 'Corporation_Z', 0)
E|||<num_to('A', 'Corporation_Z', 0)
C|||>num_option('A', 'Corporation_Z', _140)
```

Does A own an option (or an option to acquire an option) to acquire the stock in Corporation_Z? n

```
R|||>num_options('A', 'Corporation_Z', 0)
E|||<num_options('A', 'Corporation_Z', 0)
E|||<num_to('Corporation_Y', 'Corporation_Z', 50)
C|||>num_options('Corporation_X', 'Corporation_Z', _29)
```

Does Corporation_Y own an option (or an option to acquire an option) to acquire the stock in Corporation_Z? n

```
R|||>num_options('Corporation_Y', 'Corporation_Z', 0)
E|||<num_options('Corporation_Y', 'Corporation_Z', 0)
E|||<num_con_own('Corporation_Y', 'Corporation_Z', 50)
```

The number of shares of Corporation_Z stock actually owned by Corporation_Y is 0.

The number of shares of Corporation_Z stock constructively owned by Corporation_Y is 50.

Do you want to try another Client? n

End Listing Three

Listing Four

```
% THE SECTION 318(a) PROLOG PROGRAM WHICH ANSWERS HOW, HELP AND
% WHY
% By Dean A. Schlobohm
% (c) Copyright 1984 by Dean A. Schlobohm. All Rights Reserved.
% Version 1.0
% Date: 6/11/84
% Bugs:
% 1. Only includes the Section 318(a)(1) family attribution
% rules.
% 2. Very crude English interface.
% To Use: type "prolog sec318-1.pro<RET>" then after Prolog is
% loaded type "section318!<RET>"

% preliminary functions

append([],L,L) :- !.
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

appendall([],[]).
appendall([H],H).
appendall([H1,H2],L) :- append(H1,H2,L).
appendall([H1,H2|T],L) :-
    append(H1,H2,X),
    appendall(T,Z),!,
    append(X,Z,L).

member(X,[X|_]) :- !.
member(X,[_|Y]) :- member(X,Y).

add(X) :- X,!.
add(X) :- asserta(X),!.

% The Interface Program.

section318 :-
    prompt(Old,' '),
    nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,
    print(' THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES
    OF'),
    print('STOCK A CLIENT CONSTRUCTIVELY OWNS AS A RESULT OF
    SECTION '),
    print('318(a)(1) OF THE INTERNAL REVENUE CODE. '),
    nl,nl,
    ask(' Enter the name of the Client. (One word) ',
    ,Client),nl,
    ask(' Enter the name of the Corporation. (One word) ',
    ,Corp),nl,
    add(corporation(Corp)),!,
    num_family(Client,Corp,N_con,[],H)
    ,!,nl,nl,
    print(' The number of shares of ',Corp,' stock
    constructively'),
    print('owned by ',Client,' as a result of Section 318(a)(1)
    is ',N_con,'. '),
    nl,nl,!,
    ask(' Do you want to know how I reached this conclusion? ',
    ,Z),!,
    ((Z = y) ; (Z = 'Y')),nl,
    how(num_family(Client,Corp,N_con,W,H)),nl,nl,
    prompt(_ ,Old).

individual(Client,W,H) :-
    prin(' Is ', Client, ' an individual? '),
    ratom(X),nl,!,
    respond1(X,individual(Client,W,H),W).

respond1(y,individual(Client,_,[H]),_) :-
    concat([' FACT: ',Client,' is an individual.'],H),
    asserta(ind(Client,_,[H])),!.
respond1(why,individual(Client,W,H),[]) :-
    print('This is what the Internal Revenue Code provides!'),!,
    nl,
```

(Continued on next page)

Listing Four

```
individual(Client,[],H),!.
respondl(why,individual(Client,W,H),[P|T]) :-
    print(P),nl,!,
    individual(Client,T,H).
respondl(help,Y,Z) :-
    print('Section 318(a)(1) only applies to individuals.'),!,nl,
    Y.
respondl(n,individual(Client,_,[H]),_) :-
    concat(['      FACT: ',Client,' is not an individual.'],H),
    asserta(non_ind(Client,_,[H])),!,fail.
respondl(X,Y,Z) :-
    print('I do not understand! Enter y, n, help, or
    why.'),!,nl,
    Y.

% the rule for actual ownership
num_a_own(C,C,0,W,[H]) :- !.
num_a_own(Client,Corp,N,W,[H]) :-
    print('      How many shares of ',Corp,' stock does ',Client),
    print('actually own? '),
    ratom(X),nl,
    integer(X),
    concat(['      FACT: ',Client,' actually owns ',X,'
    shares.'],H),
    N = X,!.

num_a_own(Client,Corp,N,W,[H]) :-
    print('      Error -- You must enter an integer!'),nl,!,
    num_a_own(Client,Corp,N,W,[H]).

% the rule of Section 318(a)(1)
num_family(Client,Corp,N,W,H) :-
    get_rule(1,Z),
    append(W,[Z],X),
    individual(Client,X,Y),!,
    spouse_own(Client,Corp,N1,X,Q),
    family_own(Client,Corp,N2,X,R),
    N is N1 + N2,
    appendall([Z],Y,Q,R),H),!.
num_family(Client,_,0,_,H) :-
    get_rule(1,Z),
    non_ind(Client,_,Y),!,
    append([Z],Y,H),!.

% definition of spouse_own
spouse_own(Client,Corp,N,W,H) :-
    get_rule(2,Z),
    append(W,[Z],A),
    spouse(Client,L1,A,H1),
    append([Z],H1,H2),
    ind_own(Client,Corp,L1,N,H2,H3),
    appendall([Z],H1,H3),H),!.
spouse_own(Client,_,0,_,[H]) :-      concat(['      FACT: ',Client,'
    has no spouse.'],H),!.

% definition of family_own
family_own(Client,Corp,N,W,H) :-
    get_rule(3,Z),
    append(W,[Z],A),
    family(Client,L1,A,H1),
    append([Z],H1,H2),
    ind_own(Client,Corp,L1,N1,H2,H3),!,
    family_own(Client,Corp,N2,W,H4),!,
    N is N1 + N2,
    appendall([Z],H1,H3,H4),H),!.
family_own(Client,_,0,_,[H]) :-
    concat(['      FACT: ',Client,' has no more family
    members.'],H),!.

% definition of ind_own
% note Section 318(a)(5)(B) is included since ind_own does
% not include num family.
% does NOT include the other attribution rules!
```



```

ind_own(Client,Corp,[],0,_,[]):- !.
ind_own(Client,Corp,[P],N,W,H) :-
    P /= Client,!,
    num_a_own(P,Corp,N,W,H1),
    !,
    H = H1,!.
ind_own(Client,Corp,_,0,W,W) :- !.

% the definition of a family for purposes of Section 318

spouse(C,[L],W,[H]) :-
    prin('    Does ',C,' have a spouse? '),
    ratom(X),nl,
    respond2(X,spouse(C,[L],W,[H]),W),!.

spouse1(C,[L],_,[H]) :-
    prin('    What is the name of the spouse? '),
    ratom(L),nl,
    concat(['    FACT: ',L,' is the spouse of ',C,'.'],H),!.

family(C,[L],W,[H]) :-
    prin('    Does ',C,' have a child, grandchild or parent? '),
    ratom(X),nl,
    respond2(X,family(C,[L],W,[H]),W),!.

family1(C,[L],_,[H]) :-
    prin('    What is the name of the child, grandchild or
    parent? '),
    ratom(L),nl,
    concat(['    FACT: ',L,' is a family member of
    ',C,'.'],H),!.

respond2(y,Relation(Client,Person,_,H),_) :-
    concat([Relation,'l'],Z),
    Z(Client,Person,_,H),!.
respond2(why,Relation(Client,Person,_,H),[]) :-
    print('This is what the Internal Revenue Code provides!'),!,
    nl,
    Relation(Client,Person,[],H),!.
respond2(why,Relation(Client,Person,_,H),[P|T]) :-
    print(P),nl,!,
    Relation(Client,Person,T,H).
respond2(help,spouse(Client,Person,W,H),_) :-
    print('A spouse does not include one who is legally separated
    from'),
    print(Client,' under a decree of divorce or separate
    maintenance. '),
    print('Section 318(a)(1)(A)(i). '),
    !,nl,
    spouse(Client,Person,W,H).
respond2(help,family(Client,Person,W,H),_) :-
    print('A child includes a legally adopted child. Section
    318(a)(1)(B). '),
    !,nl,
    family(Client,Person,W,H).
respond2(n,Relation(Client,_,_,[H]),_) :- !,fail.
respond2(X,Y,Z) :-
    print('I do not understand! Enter y, n, help, or
    why. '),!,nl,
    Y.

% definition of rules to use in how and why

get_rule(1,'An individual is deemed to own the stock owned by
    his family. Section 318(a)(1). ').
get_rule(2,'A spouse is a family member. Section
    318(a)(1)(A)(i). ').
get_rule(3,'A child, grandchild or parent
    is a family member. Section 318(a)(1)(A)(ii). ').

% definition of "how"

how(num_family(Client,Corp,N,_,[H|T])) :-
    print('    START OF EXPLANATION'),
    nl,
    print('    The number of shares of ',Corp,' stock
    constructively owned'),
    print('by ',Client,' as a result of Section 318(a)(1) is '
    ,N,', '),
    print('determined as follows:'),nl,
    print(H),!,

```

(Continued on next page)

Tax Advisor (Listing Continued, text begins on page 64)

Listing Four

```
howl(T),!.
how(_):-
    print('      I can not give you an explanation!'),!.
howl([]):-
    nl,
    print('          END OF EXPLANATION'),
    !.
howl([H|T]):-
    nl,
    print(H),
    howl(T),
    !.
```

End Listing Four

Listing Five

THIS PROGRAM WILL DETERMINE THE NUMBER OF SHARES OF STOCK A CLIENT CONSTRUCTIVELY OWNS AS A RESULT OF SECTION 318(a)(1) OF THE INTERNAL REVENUE CODE.

Enter the name of the Client. (One word) Dean Schlobohm

Enter the name of the Corporation. (One word) Corporation

Is Dean_Schlobohm an individual? why

An individual is deemed to own the stock owned by his family.
Section 318(a)(1).

Is Dean_Schlobohm an individual? why

This is what the Internal Revenue Code provides!

Is Dean_Schlobohm an individual? y

Does Dean_Schlobohm have a spouse? why

An individual is deemed to own the stock owned by his family.
Section 318(a)(1).

Does Dean_Schlobohm have a spouse? why

A spouse is a family member. Section 318(a)(1)(A)(i).

Does Dean_Schlobohm have a spouse? why

This is what the Internal Revenue Code provides!

Does Dean-Schlobohm have a spouse? help

A spouse does not include one who is legally separated from Dean Schlobohm under a decree of divorce or separate maintenance.
Section 318(a)(1)(A)(i).

Does Dean_Schlobohm have a spouse? y

What is the name of the spouse? Mary Schlobohm

How many shares of Corporation stock does Mary_Schlobohm actually own? 300

Does Dean_Schlobohm have a child, grandchild or parent? why

An individual is deemed to own the stock owned by his family.
Section 318(a)(1).

Does Dean_Schlobohm have a child, grandchild or parent? why

A child, grandchild or parent is a family member. Section 318(a)(1)(A)(ii).

(Continued on page 92)

**UNPARALLELED
PERFORMANCE
and
PORTABILITY
in an
ISAM PACKAGE
at an
UNBEATABLE
PRICE**



c-tree
BY FAIRCOM

2606 Johnson Drive
Columbia MO 65203

The company that introduced micros to B+ Trees in 1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B+ Tree based file handlers. With c-tree™ you get:

- complete C source code written to K&R standards of portability
- high level, multi-key ISAM routines and low level B+ Tree functions
- routines that work with single-user and network systems
- no royalties on application programs

\$395 COMPLETE

Specify format:

5¼" PC-DOS 3½" Mac
8" CP/M® 8" RT-11

for VISA, MC or COD orders, call
1-314-445-6833

Access Manager and CP/M are trademarks of Digital Research, Inc.
c-tree and the circular disc logo are trademarks of FairCom

© 1984 FairCom

Circle no. 37 on reader service card.

Pascal and C Programmers

**Your programs can
now compile the
FirstTime™**

FirstTime is an intelligent editor that knows the rules of the language being programmed. It checks your statements as you enter them, and if it spots a mistake, it identifies it. *FirstTime* then positions the cursor over the error so you can correct it easily. *FirstTime* will identify all syntax errors, undefined variables, and even statements with mismatched variable types. In fact, any program developed with the *FirstTime* editor will compile on the first try.

More than a syntax checker!

FirstTime has many unique features found in no other editor. These powerful capabilities include a zoom command that allows you to examine the structure of your program, automatic program formatting, and block transforms.

If you wish, you can work even faster by automatically generating program structures with a single key-stroke. This feature is especially useful to those learning a new language, or to those who often switch between different languages.

Other Features: Full screen editing, horizontal scrolling, function key menus, help screens, inserts, deletes, appends, searches, and global replacing.

Programmers enjoy using *FirstTime*. It allows them to concentrate on program logic without having to worry about coding details. Debugging is reduced dramatically, and deadlines are more easily met.

FirstTime for PASCAL	\$245
FirstTime for C	\$295
Microsoft PASCAL Compiler	\$245
Microsoft C Compiler	\$395
Demonstration disk	\$25

Get an extra **\$100 off** the compiler when it is purchased with **FirstTime**.
(N.J. residents please add 6% sales tax.)

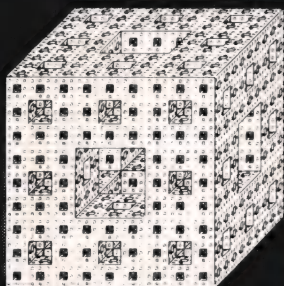
Spruce
Technology Corporation
110 Whispering Pines Drive
Lincroft, N.J. 07738
(201) 741-8188 or (201) 663-0063

Dealer enquiries welcome. Custom versions for computer manufacturers and language developers are available.

FirstTime is a trademark of Spruce Technology Corporation.



Circle no. 65 on reader service card.



WALTZ LISP^(TM)

The one and only **adult** Lisp system for CP/M users.

Waltz Lisp is a very powerful and complete implementation of the Lisp programming language. It includes features previously available only in large Lisp systems. In fact, Waltz is substantially compatible with Franz (the Lisp running under Unix), and is similar to MacLisp. Waltz is perfect for Artificial Intelligence programming. It is also most suitable for general applications.

Much faster than other microcomputer Lisps. • Long integers (up to 611 digits). Selectable radix • True dynamic character strings. Full string operations including fast matching/extraction. • Flexibly implemented random file access. • Binary files. • Standard CP/M devices. • Access to disk directories. • Functions of type lambda (expr), lambda (fexpr), lexpr, macro. • Splicing and non-splicing character macros. • User control over all aspects of the interpreter. • Built-in prettyprinting and formatting facilities. • Complete set of error handling and debugging functions including user programmable processing of undefined function references. • Virtual function definitions. • Optional automatic loading of initialization file. • Powerful CP/M command line parsing. • Fast sorting/merging using user defined comparison predicates. • Full suite of mapping functions, iterators, etc. • Assembly language interface. • Over 250 functions in total. • The best documentation ever produced for a micro Lisp (300+ full size pages, hundreds of illustrative examples).

Waltz Lisp requires CP/M 2.2, Z80 and 48K RAM (more recommended). All common 5" and 8" disk formats available.

ROCODE^(TM)
INTERNATIONAL

15930 SW Colony Pl.
Portland, OR 97224

Unix* Bell Laboratories.
CP/M* Digital Research Corp.

Version 4.4

(Now includes Tiny Prolog written in Waltz Lisp.)

\$169*

*Manual only: \$30 (refundable with order). All foreign orders: add \$5 for surface mail, \$20 for airmail. COD add \$3. Apple CP/M and hard sector formats add \$15.

Call free **1-800-LIP-4000** Dept. #11
In Oregon and outside USA call 1-503-684-3000

Circle no. 73 on reader service card.

Tax Advisor (Listing Continued, text begins on page 64)
Listing Five

Does Dean_Schlobohm have a child, grandchild or parent? why

This is what the Internal Revenue Code provides!

Does Dean_Schlobohm have a child, grandchild or parent? help

A child includes a legally adopted child. Section 318(a)(1)(B).

Does Dean_Schlobohm have a child, grandchild or parent? y

What is the name of the child, grandchild or parent?

Dean F Schlobohm

How many share of Corporation stock does Dean_F_Schlobohm actually own? 100

Does Dean_Schlobohm have a child, grandchild or parent? n

The number of shares of Corporation stock constructively owned by Dean_Schlobohm as a result of Section 318(a)(1) is 400.

Do you want to know how I reached this conclusion? y

START OF EXPLANATION

The number of shares of Corporation stock constructively owned by Dean_Schlobohm as a result of Section 318(a)(1) is 400, determined as follows:

An individual is deemed to own the stock owned by his family. Section 318(a)(1).

FACT: Dean_Schlobohm is an individual.

A spouse is a family member. Section 318(a)(1)(A)(i).

FACT: Mary_Schlobohm is the spouse of Dean_Schlobohm.

FACT: Mary_Schlobohm actually owns 300 shares.

A child, grandchild or parent is a family member. Section 318(a)(1)(A)(ii).

FACT: Dean_F_Schlobohm is a family member of Dean_Schlobohm.

FACT: Dean_F_Schlobohm actually owns 100 shares.

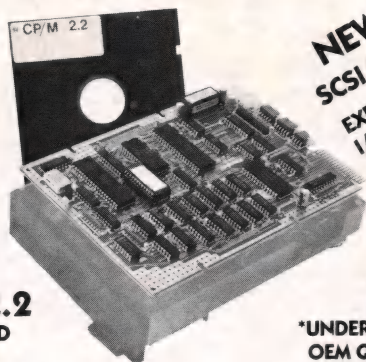
FACT: Dean_Schlobohm has no more family members.

END OF EXPLANATION

End Listings

The Little Board™ ...\$349*

The world's simplest and least expensive CP/M computer



NEW
SCSI/PLUS™
EXPANSION
I/O OPTION

CP/M 2.2
INCLUDED

***UNDER \$200 IN**
OEM QUANTITIES

- 4 MHz Z80A CPU, 64K RAM, Z80A CTC, 2732 Boot ROM
- Mini/Micro Floppy controller (1-4 Drives, Single/Double Density, 1-2 sided, 40/80 track)
- Only 5.75 x 7.75 inches, mounts directly to a 5 1/4" floppy drive
- 2 RS232C Serial Ports (75-9600 baud & 75-38,400 baud), 1 Centronics Printer Port
- Power Requirement: +5VDC at .75A; +12VDC at .05A/On-board -12V converter
- CP/M 2.2 BDOS • ZCPR3 CCP • Enhanced AMPRO BIOS
- AMPRO Utilities Included:
 - read/write to more than 2 dozen other formats (Kaypro, Televideo, IBM CP/M86....)
 - format disks for more than a dozen other computers
 - menu-based system customization
- BIOS and Utilities Source Code Available
- SCSI/PLUS Adapter:
 - Mounts directly to Little Board • Slave I/O board control • Full ANSI X3T9.2
 - 16 bidirectional I/O lines • \$99/Quantity 1

AMPRO
COMPUTERS, INCORPORATED

Distributor/Dealer/Reps
Inquiries Invited

Z80A is a registered trademark of Zilog, Inc.
CP/M is a registered trademark of Digital Research

67 East Evelyn Ave. • Mountain View, CA 94041 • (415) 962-0230 • TELEX 4940302

Circle no. 7 on reader service card.

A Professional Quality Z80/8080/8085 Disassembler

WHEN YOU NEED SOURCE FOR YOUR CODE

you need REVAS 3

REVAS interactively helps you:

Analyse your software for modification
disassemble files as large as 64K

Assign Real labels in the disassembly

Insert COMMENTS in the disassembly

Generate a Cross Reference (XREF) listing

A 60 page manual shows how the powerful **REVAS** command set gives you instant control over I/O to files, printer, or console; how to do a disassembly; and even how the disassembler works! You get on line help, your choice of assembler mnemonics, control of data interpretation, and calculation in any number base!

REVAS runs in Z80 CPM computers; is available on 8" SSSD (standard), RAINBOW, and other (ask) formats

Price: \$90.00 (plus applicable tax), Manual only: \$15.00

REVASCO

6032 Chariton Ave., Los Angeles, CA 90056

Voice: (213) 649-3575 Modem: (213) 670-9465

Circle no. 80 on reader service card.

2 Megabyte SemiDisk!

Have you been waiting on your slow floppy disk drives too long? SemiDisk Systems has a disk emulator for you! It'll put you in the fast lane, with ultra-fast data transfer, huge storage capacity, convenient battery backup, and a handy print spooler.

Have you been waiting for a SemiDisk big enough to handle your large applications programs, files, and databases - all at once? Your wait is over. SemiDisk Systems is now delivering 2 megabytes of disk storage on a single board!

512k, 1Meg and 2Megabyte SemiDisks are available for S-100 computers, (including the H/Z-100 operating under Z-DOS), IBM PC, XT, & AT, the TRS-80 Models II, 12, & 16, and the Epson QX-10. Once you've tried a SemiDisk you'll know why we say. . .

Someday you'll get a SemiDisk.

Until then, you'll just have to wait.

	512K	1Mbyte	2Mbyte
SemiDisk I, S-100	\$995	\$1795	
SemiDisk II, S-100	\$1295	\$2095	\$2549
IBM PC, XT, AT	\$945	\$1795	\$2499
QX-10, QX-16	\$799		\$2499
TRS-80 II, 12, 16	\$995	\$1795	\$2499
Battery Backup Unit	\$150		

SEMIDISK

SemiDisk Systems, Inc.

P.O. Box GG, Beaverton, Oregon 97075

503-642-3100

Call 503-646-5510 for CBBS/NW, 503-775-4838 for CBBS/PCS, and 503-649-8327 for CBBS/Aloha, all SemiDisk-equipped computer bulletin boards (300/1200 baud). SemiDisk, SemiSpool trademarks of SemiDisk Systems.

Circle no. 85 on reader service card.

We'll give your software a chance to sell itself.

We're building a "demo data base" of available software for every retailer to access. We want to add yours to it.

There are 30,000 software business packages on the market today - and new ones coming out every day. There isn't a retail computer store in America that can carry, let alone properly demo, them all. That is, up to now!

We've Got the Right Connections to Demo Your Software

We're giving every retailer in the country an opportunity to access and demo every piece of software available, provided it's in our unique data base. To get in touch with your software, he merely dials into our system using any operating computer in his store. He screens through the menu of available software packages within the customer's area of interest. The Program Finder displays information about you, your prices and how to order your product.

Lousy-Demo Insurance

Once a specific program has been selected for more information, a concise description of that

package is shown. The most important part follows. An effective demonstration of your software product prepared by you, presenting your product's best features in your own words - it's like having lousy-demo insurance.

Us In the Middle Will Solve the Muddle

If you're a retailer, the Program Finder will enable you to offer thousands of software packages without the cost of maintaining a demo library. That frees up your salespeople to spend time selling products instead of wasting time learning how to demo them.

If you're a publisher, the Program Finder will take your software public, immediately. We'll give you instant awareness in hundreds of retail stores coast to coast. And that will mean instant sales.

If you're interested, call Mitch Kolesaire, Director of Sales, at Software Information Systems, today. He'll give you the whole story. It's just what the industry has been looking for. Call him at 201-882-9141.



A product of Software Information Systems, Inc.

Finally, a New DBMS Technology

INFORMA is what **NETWORKING** is all about:
INTERACTIVE REAL-TIME DATASHARING

The experts say . . .

Corvus Systems, Inc.

"INFORMA is one of the finest multi-user Database Management Systems available for the OMNINET™ Local Area Network."

Sid Arora, Third Party Marketing Manager

TeleVideo Systems, Inc.

"INFORMA is one of the finest, true multi-user Database Management Systems we have seen run on the TeleVideo Personal Mini.™"

Mark Calkins, Product Marketing Manager

Novell

"Many of our Netware end users have found INFORMA to be a very powerful and versatile Database Management System."

Rob Walton, Manager of Independent Software Development

3COM Corporation

"The INFORMA DBMS is one of the best examples of the benefits users achieve with multi-user network software."

Robert Buchanan, Jr., Software Product Manager

•FAST •POWERFUL •EASY TO USE

•10 Level Security

•50 Keys (indexes) per record

•8000 fields per record

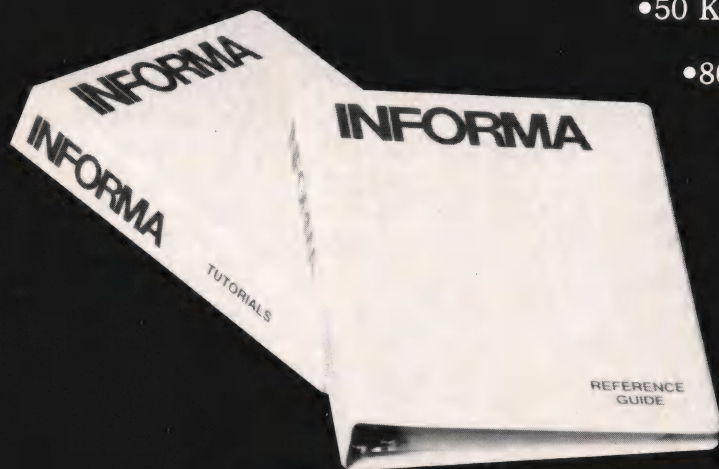
•1 sec. access from 35,000 record file

•255 screens per record

•Unlimited math and relational operations

•Intuitive "Query by Example"

•Full Formatting Reporter



**UNLIMITED PROCESSING
INCORPORATED**

8382 Baymeadows Road, Suite 8
Jacksonville, Florida 32216
(904) 731-8330 (800) 874-8555
Telex 350754 (800) 874-4185

Incredible Introductory Offer

Single-user

\$199

regularly \$795

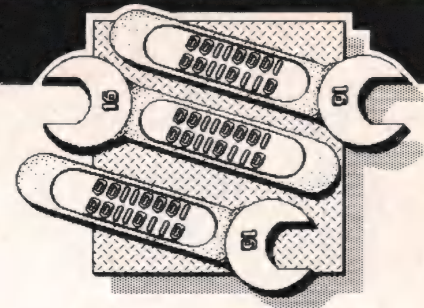
LAN/Multi-user

\$599

regularly \$1495

Available on over 20 operating systems including IBM's new PC NETWORK

16-BIT SOFTWARE TOOLBOX



by Ray Duncan

Z-8001 CPU for S-100 Bus

Yes, gentle reader, we occasionally *do* discuss some other kind of microprocessor besides the Intel family. This month, we'd like to announce the availability of an S-100 bus Z-8001 CPU card called the Super Z-10 from Way Engineering. The CPU runs at 10 MHz, is fully compatible with the IEEE 696 standard, and includes two on-board RS-232 ports, 32K of phantom EPROM, and a complete Forth monitor in EPROM. You can use the Z-8001 CPU card with Compupro high-speed static RAM boards and with most popular S-100 hard disk controllers. The introductory cost of the Super Z-10 CPU is \$1000.00.

Digital Research CP/M-8000 is available in a customized version for the Super Z-10 and includes the usual CP/M utilities (PIP, ED, DDT, and so on) as well as a C compiler, assembler, and linker. The BIOS is written in C and assembly language, and the source code is supplied. A disk-based Forth development system that runs under CP/M-8000 will also be available in the near future. CP/M-8000, which requires a hard disk and 256K of RAM, costs \$350.00. For more information, contact Way Engineering at 2011 Tulip Tree Lane, La Canada, CA 91011 (213) 245-1480. I am taking delivery of one of these systems within the next few days, so you should see a detailed report in a later column.

MacFeedback

Mike Cohen of the Bronx writes: "In the December 1984 issue, you incorrectly state that the Macintosh 68000 assembler requires two machines. It simply isn't true, as I've used that assembler along with Consulair's excellent Mac C compiler on a single Macintosh with no problems. True, Apple

provides a debugger, which does require two machines, but the Macsbug debugger (which is also provided on the same disk) requires only one machine. It does have some problems in 128K, since it can't remain resident while the assembler or editor are being used. There are no such problems on a 512K Macintosh, since I keep the full screen debugger resident while editing, assembling, and compiling programs.

"Also, the statement that most high-level languages are 'incredibly weak, bug-ridden, and slow' is untrue of everything other than MS-Basic. MacPascal is one of the best versions of Pascal I've seen on any micro and reasonably fast—it can calculate 1500 Factorial (using extended precision math) in an amazing 10 seconds. Consulair Corp.'s Mac C compiler is fast, powerful, and bug-free. It's reasonably Unix-compatible, although using the Mac Toolbox library rather than STDLIB results in much smaller applications (a text editor can be easily done in less than 4K).

"To get things started with 68000 software, here are two functions I wrote for use with the Mac C compiler to provide the standard file 'Open' and 'Save as' dialogs. Most of the code was output directly from Mac C (particularly the in-line trap instructions and parameter saving).

"The function OldFileName(Count, typeList, Reply) will display the open dialog, which lists all files of the specified type. NewFileName(DefaultName, Reply) will display the save dialog, with the default filename displayed for editing. Both functions return a zero if the request was cancelled, otherwise returning nonzero, and Reply (pointer to struct SFReply, as defined in Inside Macintosh and packages.h supplied with Mac C) will contain the filename as volume reference number of the file selected." See Listing One (page 102)

for Mr. Cohen's contribution.

Well, having tried or observed friends being tried by MacPascal, MacBasic, and MacForth, I'll stand by my earlier statements about those languages at least. They don't meet my criteria for "insanely great." On the Apple assembler, I was in error. I based my previous column on Apple's own product announcement, which clearly stated "two machines required," and had not received the actual software when I submitted the column to *DDJ* for publication. The Macsbug debugger, by the way, is insanely clever. When loaded, it simply requests some space on the heap and then hides itself there like any other data object, grabbing back control when an exception is detected.

More Mac News

As far as program development tools go for the Mac, there is hope for the rest of us after all. Mainstay of Agoura Hills, CA, has announced MacASM, a macroassembler for the Macintosh with an integral full-screen editor and resource compiler that costs only \$125.00. I already have received several favorable letters from *DDJ* readers about this product. You can contact Mainstay at (818) 991-6540.

But the neatest news of all is the impending release of Neon from Kriya Systems, which we recently saw demonstrated at the Las Vegas Comdex Apple booth. This is a high-level, object-oriented language, a la Smalltalk, that will sell for only \$150.00 and allows full access to the Macintosh's Toolbox and graphics capabilities. Best of all, the package includes extensive source code, a 600-page manual, and a toll-free license to distribute Neon embedded in application programs. Based on a 32-bit implementation of Forth, Neon makes available to

the programmer the full spectrum of Forth commands and extensibility. The scheduled release date of Neon is February 15, so if all goes well, it should be available by the time you read this. You can reach Kriya at 505 N. Lake Shore Drive, Suite 5510, Chicago, IL 60611.

At the other end of the spectrum, we have the advertisements heralding "CP/M for the Macintosh" from IQ Software in Fort Worth, TX. I'll bet this advertisement made the Apple software gurus gnash their teeth and pull their hair out! I can't imagine anything more antithetical to the whole MacMiliu than DIR, PIP, ED, and STAT.

80286 Opcode of the Month

One of the opcodes added in the 80286 was the instruction called BOUNDS, which checks whether a number falls within limits and generates a hardware interrupt 5 if it does not. This nifty opcode was added for the sake of compiler writers who wanted an efficient way to check the validity of an array index at runtime. Don't rush to use this instruction in your assembly language programming experiments on the IBM PC/AT, however, or you may get an unpleasant surprise.

It seems that way back when the PC/AT was just a gleam in the eye of the president of the Entry Systems Division (in fact, no such division even existed then), somebody at either IBM or Microsoft assigned interrupt 5 to the Print Screen ROM BIOS routine. So, when the 80286 executes a BOUNDS instruction, detects a value outside of the specified range, and executes interrupt 5... you guessed it! To add insult to injury, the 80286 assumes that the interrupt 5 handler will fix up the offending value, so it executes the BOUNDS instruction again. Of course, the ROM BIOS program to print the screen carefully saves and restores all the registers, including the one containing the nasty value that caused the interrupt, because it doesn't know anything about BOUNDS exceptions or fix ups or anything else for that matter. The result: one screen dump after another until you give up and restart the computer.

IBM has distributed a fix for this problem in its Software Support Center news bulletin. The fix involves installa-

Debugging Bugging You?

Torpedo program crashes and debugging delays with debugging dynamite for the IBM PC ...

UP PERISCOPE!

First, you install the hardware.

The hardware's a special memory board that fits in a PC expansion slot. Its 16K of write-protected memory contains Periscope's resident symbolic debugger. No runaway program, however berserk it may be, can touch this memory!

Then you UP PERISCOPE.

Use Periscope's push-button break-out switch to interrupt a running program ... even when the system's hung! Periscope supports Assembly, BASIC, C and Pascal. In addition to the usual debugging capabilities, some of Periscope's features are:

Stop your system in its tracks at any time.

Use symbol names instead of addresses.

Run a program on one monitor and debug on another.

Monitor your program's execution with Periscope's comprehensive breakpoints.

Debug memory-resident programs.

Put your time to better use.

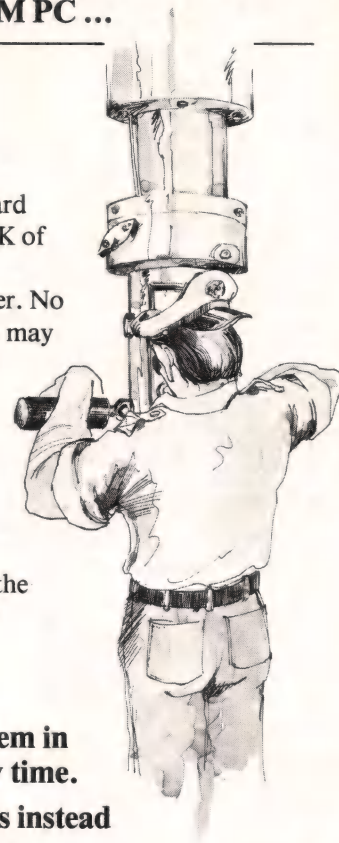
The Periscope system is \$295. It carries a 30-day money-back guarantee and includes the memory board, remote break-out switch, debugger software, 100-page manual, and quick-reference card. The memory board is warranted for one year. A demonstration disk is \$5.00.

System requirements for Periscope are an IBM PC, XT or Compaq, PC-DOS, 64K RAM, 1 disk drive and an 80-column monitor. For MasterCard and Visa orders only, call 800/421-5300 (ext. R96) 24 hours a day. For additional information, call 404/256-3860 from 9 AM to 5 PM Eastern Time.

Get your programs up and running;

UP PERISCOPE!

Data Base Decisions / 14 Bonnie Lane / Atlanta, GA 30328



#69 Volume VII, No. 7:

IBM-PC Issue: CP/M-86 vs. MSDOS (A Technical Comparison)—Hi-Res Graphics on the IBM-PC—PDP-1802, Part II—Review of Word Processors for IBM.

#70 Volume VII, No. 8:

Argum "C" Command Line Processor—SEND/RECEIVE File Transfer Utilities—Intel's 8087 Performance Evaluation.

#71 Volume VII, No. 9:

FORTH Issue: Floating-Point Package—H-19 Screen Editor—Relocating, Linking Loader—Z8000 Forth—Forth Programming Style—8086 ASCII-Binary Conversion Routines—CP/M Conditional SUBMIT.

#72 Volume VII, No. 10:

Portable Pldgin for Z80—68000 Cross Assembler, Part I—MODEM and RCPI/Ms—Simplified 68000 Mnemonics—Nested Submits—8086/88 Trig Lookup.

#73 Volume VII, No. 11:

Wildcard UNIX Filenames—Tests for Pldgin—68000 Cross Assembler Listing, Part 2—Adding More BDOS Calls—The Perfect Hash—BASIC Memory Management—Benchmarks for CP/M-86 vs. MSDOS, and the 8087.

#77 Volume VIII, Issue 3:

The Augusta P-Code Interpreter—A Small-C Operating System—6809 Threaded Code: Parametrization and Transfer of Control—A Common-Sense Guide to Faster, Small BASIC—A Fundamental Mistake in Compiler Design—Basic Disk I/O, Part I.

#78 Volume VIII, Issue 4:

RECLAIM Destroyed Directories—Binary Magic Numbers—8080 Fig-Forth Directory & File System—SAY" Forth Votrax Driver—TRS-80 8080 to Z80 Translator—Basic Disk I/O, Part II.

#79 Volume VIII, No. 5:

The Augusta Compiler—A Fast Circle Routine—Enhancing the C Screen Editor—Shifts and Rotations on the Z80—The SCB, TSX, and TXS Instructions of the 6502 and 6800—MS-DOS vs. CP/M-86—Controlling MBASIC—The Buffered Keyboard—IBM PC Character Set Linker—Flip Utility for the IBM PC.

#80 Volume VIII, Issue 6:

Fast Divisibility Algorithms—B-Tree ISAM Concepts—CP/M BDOS AND BIOS Calls for C—Serial Expansion in Forth—Fast Matrix Operations in Forth, Part I—Yes, You Can Trace Through BDOS—Julian Dates for Microcomputers—8088 Addressing Modes—8088 Line Generator—CP/M Plus.

#81 Volume VIII, Issue 7:

The Augusta Compiler, continued—RED: A Better Screen Editor, Part I—Anatomy of a Digital Vector and Curve Generator—Fast Matrix Operations in Forth, Part II—The AGGH-HHI Program—MBOOT Revisited—CP/M Plus Feedback—MS-DOS Rebuttal—68000 Tools—Sizing Memory on the IBM PC.

#82 Volume VIII, Issue 8:

Serial-to-Parallel: A Flexible Utility Box—McWORDER: A Tiny Text Editor—And Still More Fifth Generation Computers—Specialist Symbols and I/O Benchmarks for CP/M Plus—CP/M Plus Memory Management—Zero Length File Test—PAUSEIF, QUITIF, and now SKIPF—ACTxx Cross Assemblers.

#83 Volume VIII, No. 9:

FORTH ISSUE: Forth and the Motorola 68000—Nondeterministic Control Words in Forth—A 68000 Forth Assembler—GO in Forth—Precompiled Forth Modules—Signed Integer Division—Some Forth Coding Standards—The Forth Sort.

#84 Volume VIII, No. 10:

Unix to CP/M Floppy Disk File Conversion—A Small-C Help Facility—Attaching a Winchester Hard Disk to the S-100 Bus—Using Epson Bit-Plot Graphics—8086/88 Function Macros—Auto Disk Format Selection—CP/M Plus Device Tables.

#85 Volume VIII, Issue 11:

A Kernel for the MC68000—A DML Parser—Towards a More Writable Forth Syntax—Simple Graphics for Printer—Floating-Point Benchmarks.

#86 Volume VIII, Issue 12:

Faster Circles for Apples—Cursor Control for Dumb Terminals—Dysan's Digital Diagnostic Diskette—Interfacing a Hard Disk Within a CP/M Environment—The New MS-DOS EXEC Function.

#87 Volume IX, Issue 1:

A Structured Preprocessor for MBASIC—A Simple Window Package—Forth to PC-DOS Interface—Sorted Diskette Directory Listing for the IBM PC—Emulate WordStar on TOPS-20—More on optimizing compilers—The PIP mystery device contest.

#88 Volume IX, Issue 2:

Telecommunications Issue: Micro to Mainframe Connection—Communications Protocols—Unix to Unix Network Utilities—VPC: A Virtual Personal Computer for Networks—PABX and the Personal Computer—BASIC Language Telecommunications Programming—U.S. Robotics S-100 Card Modem.

#89 Volume IX, Issue 3:

RSA: A Public Key Cryptography System, Part I—Introduction to PLUC: Programming Language for Compilers—Program Design Using Pseudocode—More on Binary Magic Numbers—How fast is CP/M Plus?—CP/M 2.2 BIOS Function: SELDSK—The results of the Floating-Point benchmark.

#90 Volume IX, Issue 4:

Optimizing Strings in C—Expert Systems and the Weather—RSA: A Public Key Cryptography System, Part II—Several items on CP/M Plus, CP/M v2.2 Compatibility—BDOS Function 10: Vastly Improved—More on MS-DOS EXEC Function—Low-Level Input-Output in C.

#91 Volume IX, Issue 5:

Introduction to Modula-2 for Pascal Programmers—Converting Fig-Forth to Forth-83—Sixth Generation Computers—A New Library for Small-C—Solutions to Quirks in dBASE II.

#92 Volume IX, Issue 6:

CP/M on the Commodore 64—dBASE II Programming Techniques—First Chinese Forth: A Double-Headed Approach—cc-A Driver for a Small-C Programming System—A New Library for Small-C (Part II)—Comments on Sixth Generation Computers—Review of Turbo Pascal.

#93 Volume IX, Issue 7:

RSX under CP/M Plus—p-A Small-C Preprocessor—A Simple Minimax Algorithm—Languages and Parentheses (A Suggestion for Forth-like Languages)—Comments on assembly language development packages, RSX to patch CP/M 2.2 with CP/M, iRMS-86 for the IBM PC, C Programming Tools.

#94 Volume IX, Issue 8:

SCISTAR: Greek and Math Symbols with WordStar—A File Comparator for CP/M Plus—Designing a File Encryption System—A Small-C Concordance Generator.

#95 Volume IX, Issue 9:

Forth Special Issue—File Maintenance in Forth—Forth and the Fast Fourier Transform—Computing with Streams—A Forth Native-Code Cross Compiler for the MC68000—The FVG Standard Floating-Point Extension—CP/M Plus: Interbank Memory Moves Without DMA—ways to make C more powerful and flexible.

#96 Volume IX, Issue 10:

More dBASE II Programming Techniques—Simple Calculations with Complex Numbers—GREP: C: A Unix-like Generalized Regular Expression Parser—An optimization scheme for compilers, MSDOS 2.0 Filters, Sizing RAM under MSDOS, Two programming systems illustrating Runge-Kutta integration.

#97 Volume IX, Issue 11:

Adding Primitive I/O Functions to muLISP—Program Monitor Package: Using Interrupts to Instrument Applications—CP/M 2.2 Goes PUBLIC—A Guide to Resources for the C Programmer—RESORT.

#98 Volume IX, Issue 12:

Varieties of Unix—Unix Device Drivers—A Unix Internals Bibliography—A file Browser Program—An Introduction to Parsing.

#99 Volume X, Issue 1:

Fatten Your Mac—QuickDraw Meets Image Writer—Archiving Files with CP/M 80 and CP/M 86—MBOOT and MODEM for the C-64's CP/M—Unstructured Forth Programming: An Introduction.

#100 Volume X, Issue 2:

Festschrift for Doctor Dobb—Fire in the Valley—Tiny Basic for the 68000—An Enhanced ADFGVX Cipher System—More dBase Tips & Techniques.

TO ORDER:

Send \$3.50 per issue to: **Dr. Dobb's Journal**, 2464 Embarcadero Way, Palo, Alto, CA 94303

Please send me the issue(s) circled: **69 70 71 72 73 78**

77 79 80 81 82 83 84 85 86 87 88 89

90 91 92 93 94 95 96 97 98 99 100

I enclose \$_____ (U.S. check or money order).

Outside the U.S., add \$.50 per issue.

I have read the postal instructions and understand that I will not receive my order unless I have sent the correct payment amount.

Please charge my: ☐ Visa ☐ MIC ☐ Amer. Exp.

Card No. _____ Exp. Date _____

Signature _____

Name _____

Address _____

City _____

State _____ Zip _____

Availability on first come/first serve basis. Outside the U.S. add \$.50 per issue ordered. Price includes issue, handling, and shipment by second class or foreign surface mail. Within the U.S., please allow 6-9 weeks to process your order second class. For faster service within the U.S., we'll ship UPS if you add \$1.00 for 1-2 issues and \$.50 for each issue thereafter. We need a street address, not a P.O. Box. Airmail rates: To Canada add \$1.75 per magazine, all other foreign add \$3.00 per magazine.

tion of a new interrupt 5 handler that does the following when it receives control:

(1) It uses the address pushed on the return stack to peek into the program that was executing to determine whether a BOUNDS opcode caused the exception.

(2) If the cause was, in fact, a BOUNDS opcode, then it performs both the appropriate fix up and an interrupt return.

(3) Otherwise, it transfers control to the Print Screen ROM BIOS routine. Not very aesthetically pleasing, but this fix seems to work.

DOS 3.0 Differences

In my last column, I discussed some of the documented differences between PCDOS 2.0 and DOS 3.0. Here's news about one I missed. The unexplained failure of a once perfectly happy program prompted me to investigate the behavior of the Find First and Find Next functions (04EH and 04FH, respectively). Much to my surprise, I found that the definition of the Find Next function has been changed in DOS 3.0 for no obvious reason. Furthermore, the change is not documented on DOS Technical Manual page vi, which is supposed to warn the programmer about new or modified DOS function calls.

Under DOS 2.0, function 4FH did not take any register arguments but assumed that the information from the previous call to Find First or Find Next was in the current DTA. Under DOS 3.0, the manual says that function 4FH requires DS:DX to "contain the information from a previous 'Find First' call." Because there's no conceivable reason why function 4FH would want the original filename string address, I interpret this rather vague wording to mean that DS:DX should point to the same buffer that you set the DTA to before calling Find First. In the case of my program at least, the problem turned out to be not function 4FH at all but the fact that function 4EH does not return register AX = 0 for a successful call as it did under DOS 2.0. Apparently, under DOS 3.0, you absolutely *must* check the carry bit first; if it isn't set, don't assume that register AX will contain

anything in particular.

Naturally, a day or two after I'd wasted the afternoon tracking down the above, I got a letter from Dan Daetwyler serving up the same information in his usual lucid way. He went on to write: "The DOS 3.0 writers finally saw fit to give us an interface to the resident PRINT function. It's described as interrupt 2FH in the manual. I had a critical need for this function, and it was one of the reasons I quickly hopped on the V3.0 bandwagon. So I read the manual and then tried to implement the function. The manual states that the AL register (and for

some calls, some other registers) must be set up before the interrupt is issued. A set of values for AL are given, with specific details of the return values, other registers, etc. Not bad!

"What's the problem? Simple. Nowhere is the AH register mentioned. But it turns out that unless AH contains a binary one at interrupt time, you will have no chance at all of getting the thing to work! I found it out by the cut and try method, and lucked out. Since then I've disassembled the PRINT module and confirmed my 'guess.' Not so incidentally, this interrupt is apparently going to be used for

CP/M-80 C Programmers . . .

Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast*, then it's

time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming.

BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/280 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.
- Plus . . .
- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner and the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group (\$10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for its remarkable speed.

BYTE Magazine placed BDS C ahead of all other 8080/280 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as

fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

"I recommend both the language and the implementation by BDS very highly."

Tim Pugh, Jr.
in *InfoWorld*

"Performance: Excellent
Documentation: Excellent
Ease of Use: Excellent"

InfoWorld

Software Report Card
"... a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

Don't waste another minute on a slow language processor. Order your BDS C Compiler today!

Complete Package (two 8" SSD disks, 181-page manual): **\$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS C is designed for use with CP/M-80 operating systems, version 2.2 or higher. It is not currently available for CP/M-86 or MS-DOS.

BD Software

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

Circle no. 12 on reader service card.

TURBO GRAPHICS™

for use with

Borlands Turbo Pascal™

A comprehensive set of Pascal procedures and binary routines that includes most of the graphics capabilities found in BASICA in addition to several new graphics routines.

MAJOR FEATURES

(on medium resolution graphics screen)

- Draw large letters, circles, and lines in color.
- Create complicated graphics using DRAW "macros".
- Create and use WINDOWS non-destructively.
- Paint enclosed areas. Get and Put screen memory.
- Scroll any portion of the screen up or down.
- Use the font editor to create letters and graphics characters.
- Combine features to create interesting animations.

Catalog Numbers:

UTIL-100-IBM PC \$39.95
UTIL-100-IBM PCjr each

Shipping and Handling:

\$5.00 each



Diversified Educational Enterprises, Inc.
725 MAIN STREET
LAFAYETTE, IN 47901
317-742-2690

Circle no. 29 on reader service card.

```
Declarations: globals
char c0;int x0;
char c,cl,ca[100];int x,xl,xa[100];
```

```
Declarations: locals
char cl,c11,cal[100];int xl,x11,xal[100];
```

Test #	Z80A 4 MHz BDS C vl.43 msec	286 6 MHz C86 vl.33 msec	Ratio	statement
--------	-----------------------------------	--------------------------------	-------	-----------

1	5	1	5.00	c='a';
2	6	1	6.00	x=1;
3	6	1	6.00	x=1234;
4	15	4	3.75	if(c)x=1;
5	15	4	3.75	if(x)x=1;
6	13	5	2.60	c <<= 1;
7	21	5	4.20	c <<= 4;
8	11	4	2.75	x <<= 1;
9	19	5	3.80	x <<= 4;
10	33	5	6.60	c >>= 1;
11	69	5	13.80	c >>= 4;
12	31	4	7.75	x >>= 1;
13	67	5	13.40	x >>= 4;
14	8	2	4.00	c=ca[0];
15	17	10	1.70	c=ca[c0];
16	8	2	4.00	x=xa[0];
17	22	6	3.67	x=xa[x0];
18	8	2	4.00	c=c0;
19	8	1	8.00	x=x0;
20	11	4	2.75	c=c+1;
21	5	3	1.67	c++;
22	9	2	4.50	x=x+1;
23	9	2	4.50	x++;
24	4	1	4.00	cl='a';
25	11	1	11.00	xl=1;
26	11	1	11.00	xl=1234;
27	20	4	5.00	if(cl)xl=1;
28	27	4	6.75	if(xl)xl=1;
29	16	5	3.20	cl <<= 1;
30	24	5	4.80	cl <<= 4;
31	25	4	6.25	xl <<= 1;
32	33	5	6.60	xl <<= 4;
33	37	5	7.40	cl >>= 1;
34	72	5	14.40	cl >>= 4;
35	46	4	11.50	xl >>= 1;
36	81	5	16.20	xl >>= 4;
37	12	2	6.00	cl=cal[0];

Test #	Z80A 4 MHz BDS C vl.43 msec	286 6 MHz C86 vl.33 msec	Ratio	statement
--------	-----------------------------------	--------------------------------	-------	-----------

38	21	10	2.10	cl=cal[c0];
39	22	3	7.33	xl=xal[0];
40	33	6	5.50	xl=xal[x0];
41	8	2	4.00	cl=c0;
42	15	2	7.50	xl=x0;
43	11	4	2.75	cl=cl+1;
44	4	3	1.33	cl++;
45	24	2	12.00	xl=xl+1;
46	17	2	8.50	xl++;
47	253	18	14.06	movmem(&ca[0],&ca[50],1);
48	274	21	13.05	movmem(&ca[0],&ca[50],5);
49	218	47	4.64	strcpy(&ca[50],&ca[0]);
				/*1 char*/
50	764	192	3.98	strcpy(&ca[50],&ca[2]);
				/*11 chars*/
51	14	14	1.00	dummy();
				/*just returns*/

Notes:

1. All times accurate to 1 microsecond.
2. Z80A tests were run at 4 MHz in an IMS International 8000 CPM80 system. 80286 tests were run at 6 MHz (no wait states) on a Lomas Data Products LDP2 MSDOS 2.1 system.
3. For BDS-C both -o and -e optimizations were used.
4. Test program generation and result reporting done with T/Maker.

Figure 1

Thomas Morgan's Benchmarks

some of the TopView or file-sharing functions, and if you issue the interrupt with the AH register containing large values (0BBH range), you end up going to a totally different function. Undocumented, of course!

"I'm getting a bit tired of getting manuals filled with good advice (it really is) on programming techniques and then finding out that the operating system violates these 'good practices' right and left. The PRINT module, for example, not only includes a significant array of self-modifying code (ugh), but it even 'relocates' (within its own area . . . why?) its interrupt service routines then does adjustment of the addresses to allow for the self-imposed relocation! (Double ugh.) I'm sure I'll eventually find out that there is a rationale for this, but I strongly suspect it will be a 'cop-out'. I've never yet found a reason for flagrant violation of good programming practices! This case seems to be caused by a desire to save a couple of hundred bytes of memory. Well, everyone knows that DOS is growing fast enough, without wasted memory, but it also appears that the memory could have been saved by simple reorganization of the source module. If I ever get the courage to complete the disassembly, I'll find out!"

Microsoft Assembler Bug of the Month

The program in Listing Two (page 102) will assemble without any errors; however, when you try to link it, you get the error message:

```
Invalid Object Module.  
Input file: TEST.OBJ(A)  
pos: 00044 Record Type: A0
```

In search of further enlightenment, we looked this message up in the DOS 3.0 manual (page A-58) and found the advice:

Explanation: LINK. Object module(s) was incorrectly formed or unobserved errors occurred during compilation. The disk may be bad.
Action: Recompile the module to either the same or a different disk.

There's no explanation at all of what

the various record types for the Linker consist of or what "pos: 00044" is supposed to mean (if you knew the latter, you could probably figure out the offending instruction regardless of the inscrutable error message).

Repeated attempts at assembly and linking, on different disks, using different versions of DOS and/or the Linker, produced the same or other confusing error messages. To make a long story short, either the Assembler or the Linker can't cope with a DD data declaration statement inside the invocation of a structure macro. Our thanks to the boys at Microsoft for yet another interesting puzzle; we trashed an entire day figuring out that this was the only problem in a 5000-line graphics driver.

Some 80286 Timings

Thomas Moran writes: "Here are some comparisons of individual C statement times using BDS C on a Z80 and C86 on an 80286. Most of the C programs I run seem to be around three times as fast on the 286 as the Z80. (They run slightly faster on the Z80 than on the IBM PC.)

"Perhaps the most interesting results are (1) the C86 code for a subroutine call takes as long to execute on the 80286 as the BDS C code on the Z80, and (2) strcpy is a horror in both cases. One often codes *p++ . . . for scanning and copying since C has no language constructs to handle strings. Perhaps a smart compiler could realize that a subroutine could be replaced by a few machine-repeated move or scan instructions." Thomas's timings are presented in the figure on page 100.

16-Bit Software Toolbox Installment

For my own code contribution this month, I'm providing two 8086 assembly language subroutines of general interest (Listing Three, page 104). The first, BIN_TO_ASC, converts a signed 32-bit integer to an ASCII string in the desired radix (the radix must be in the range 2-36). The upper part of the 32-bit value is passed in DX and the lower part in AX; the radix is passed in CX; and SI contains the last byte address of the area to store the converted

ASCII string. It is the responsibility of the user to make sure that the destination buffer is large enough for any number that might be printed in the desired radix.

DIVIDE is a general purpose 32-bit by 16-bit unsigned divide subroutine. The 32-bit dividend is passed with the upper part in DX and the lower part in AX; the divisor is passed in CX. The 32-bit quotient is returned in DX:AX, and the remainder in BX. Contrast this with the 8086's built-in unsigned DIV instruction, which also divides DX:AX by a 16-bit value but can return only a 16-bit quotient; this is inadequate for an output conversion routine that must extract digits from a 32-bit argument. The code for DIVIDE may look a little obscure at first, but it will become clearer if you simply consider each of the registers DX, AX, and CX as a single digit. Then you'll find that it works exactly like normal long division done by hand.

DDJ

(Listings begin on next page)

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 195.

**PROFESSIONAL
BASIC™**

JUST BECAME AFFORDABLE

\$99

"outstanding" Ray Duncan - Dr. Dobbs' Journal

Use 640k (e.g. 250 x 250 array)
Dynamic Syntax Checking
19 Debugging Windows
Run PC BASICA Programs

OPTIONAL: 8087/80287 Support — \$50

M Morgan Computing Co., Inc.
(214) 739-5895
10400 N. Central Expwy., Suite 210
Dallas, TX 75231

Circle no. 51 on reader service card.

Listing One

Mike Cohen's standard file "Open" and "Save" dialogs for the Macintosh

```
;Consulair Mac C Compiler 1.0
STRING_FORMAT 0
INCLUDE M68KLIB.D
OldFileName:
    CLR.L        -(SP)
    LINK         A6,#-20
    MOVEM.L      D0/D1/D2,-16(A6)
    PEA          -20(A6)
    DC.W         $A874        ;GetPort
    MOVE.W       #80,-(SP)
    MOVE.W       #80,-(SP) ;location
    CLR.L        -(SP) ;Prompt
    CLR.L        -(SP) ;FileFilter
    MOVE.W       D0,-(SP) ;numTypes
    MOVE.L       D1,-(SP) ;TypeList
    CLR.L        -(SP) ;DlgHook
    MOVE.L       D2,-(SP) ;Reply
    MOVE.W       #2,-(SP) ;routine selector for GetFile
    DC.W         $A9EA        ;Pack3
    MOVE.L       -20(A6),-(SP)
    DC.W         $A873        ;SetPort
    MOVE.L       -8(A6),A0
    MOVE.B       (A0),D0
    EXT          D0
    EXT.L        D0
    RETX
    XREF         OldFileName
;
```

```
NewFileName:
    CLR.L        -(SP)
    LINK         A6,#-16
    MOVEM.L      D0/D1,-12(A6)
    PEA          -16(A6)
    DC.W         $A874        ;GetPort
    MOVE.W       #80,-(SP)
    MOVE.W       #80,-(SP) ;location
    PEA          @1
    MOVE.L       D0,-(SP) ;OrigName
    CLR.L        -(SP) ;DlgHook
    MOVE.L       D1,-(SP) ;Reply
    MOVE.W       #1,-(SP) ;routine selector for PutFile
    DC.W         $A9EA        ;Pack3
    MOVE.L       -16(A6),-(SP)
    DC.W         $A873        ;SetPort
    MOVE.L       -8(A6),A0
    MOVE.B       (A0),D0
    EXT          D0
    EXT.L        D0
    RETX
@1: DCB          8
    DC.B         'Save As:'
    XREF         NewFileName
```

End Listing One

Listing Two

Demonstrates obscure Microsoft assembler/linker bug.

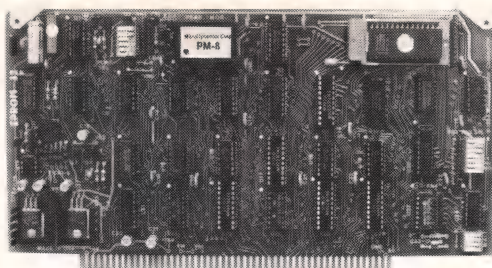
```
title    'Show Structure Bug'
page     60,132

dseg     segment para public 'DATA'
```

(Continued on page 104)

S-100 EPROM PROGRAMMER

EPROM-32



- Field-proven board meets IEEE-696 standard.
- Programs 1K through 32K (byte) EPROMs.
- Textool zero-insertion-force programming socket.
- EPROM is programmed through I/O ports and can be verified through I/O ports or located in memory space for verification.
- Programming voltage generated on-board.
- Personality Modules adapt board to EPROMs:

PM-1—2508, 2758	PM-3—2732, 2732A	PM-6—68764
2516, 2716	PM-4—2564	PM-8—27128
PM-2—2532	PM-5—2764	PM-9—27256

- Feature-packed CP/M-compatible control software includes fast programming algorithm.
- One year warranty.

\$269.95*
(A & T)

MicroDynamics Corporation

Suite 245 • 1355 Lynnfield Road • Memphis, TN 38119
(901)-682-4054

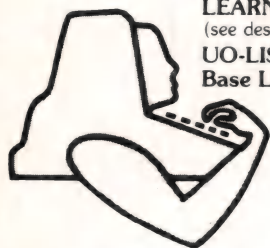
* Price includes EPROM-32, documentation and two Personality Modules (specify). Additional Modules—\$7.95. Control software on 8" SSD diskette—\$29.95. UPS ground—\$2.00. UPS air—\$4.00. COD—\$1.65. foreign add \$15.00. VISA & MASTERCARD welcome.

See Dec. 1983 **Microsystems** for a review of the EPROM-32.

Circle no. 39 on reader service card.

(LISP)

Artificial Intelligence Language UO-LISP Programming Environment The Powerful Implementation of LISP for MICRO COMPUTERS



LEARN LISP System (LLS.1) **\$39.95**
(see description below)
UO-LISP Programming Environment
Base Line System (BLS.1) **\$49.95**

Includes: Interpreter, Compiler, Structure Editor, Extended Numbers, Trace, Pretty Print, various Utilities, and Manual with Usage Examples. (BLS.1) expands to support full system and products described below.

UO-LISP Programming Environment: The Usual LISP Interpreter Functions, Data Types and Extensions, Structure & Screen Editors, **Compiler, Optimizer, LISP & Assembly Code Intermixing**, Compiled Code Library Loader, I/O Support, Macros, Debug Tools, Sort & Merge, On-Line Help, Other Utility Packages, Hardware and Operating System Access, Session Freeze and Restart, Manual with Examples expands to over 350 pages. Other UO-LISP products include: LISPTX text formatter, LITTLE META translator writing system, RLISP high level language, NLARGE algebra system. Prices vary with configurations beyond (BLS.1) please send for **FREE** catalog.

LEARN LISP System (LLS.1): Complete with LISP Tutorial Guide, Editor Tutorial Guide, System Manual with Examples, Full LISP Interpreter, On-Line Help and other Utilities. LEARN LISP fundamentals and programming techniques rapidly and effectively. This system does not permit expansion to include the compiler and other products listed above.

LISP Tutorial Support (LTS.1): Includes LISP and Structure Editor Tutorial Guides, On-line Help, and History Loop. This option adds a valuable learning tool to the UO-LISP Programming Environment (BLS.1). Order (LTS.1) for **\$19.95**.

REQUIRES: UO-LISP Products run on most Z80 computers with CP/M, TRSDOS or TRSDOS compatible operating systems. The 8086 version available soon.

TO ORDER: Send Name, Address, Phone No., Computer Type, Disk Format Type, Package Price, 6.5% Tax (CA residents only), Ship & Handle fee of \$3.00 inside U.S. & CN, \$10 outside U.S., Check, Money Order, VISA and MasterCard accepted. With Credit Card include exp. date. Other configurations and products are ordered thru our **FREE** catalog.

Northwest Computer Algorithms

P.O. Box 90995, Long Beach, CA 90809 (213) 426-1893

Circle no. 61 on reader service card.

ConIX™

UNIX™ Technology for CP/M™

ConIX can provide any 48K+ CP/M-80 compatible system with many advanced capabilities of UNIX. You'll be amazed at what your 8-bit micro can do now! ConIX features include:

I/O Redirection and Pipes (uses memory or disk), multiple commands per line, full upper/lower case and argument processing, Auto Screen Paging, Programmable Function Keys, improved User Area Directory manipulation, Command and Extension (Overlay) Path Searching, "Virtual" disk system, 8Mb Print Spooler, extensive preprocessed "Shell" command programming language, 300+ variables, over 100 built-in commands, Math Package, 22 new BDOS SysCalls, Archiver (compacts files for disk space savings of over 50%), On-Line Manual System, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs quickly and easily without any system modifications.

The ConIX Operating System List Price: \$165

Price includes Instructional Manual, 8" SSD disk, and free support. 5 1/4" format conversions available. To order, contact your local dealer or buy direct and add shipping: \$4.50 UPS, \$10 Canada, \$25 overseas COD \$2 extra (USA only). NY State residents add sales tax.



Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786

Dealer inquiries invited!

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

Circle no. 22 on reader service card.

**PRESENTING
THE
MEGAMAX C COMPILER**

FEATURING: ONE PASS
• IN-LINE ASSEMBLY • SUPPORT OF DYNAMIC
OVERLAYS • FULL ACCESS OF MACINTOSH
TOOLBOX ROUTINES • AND MUCH MORE ...
DEVELOPMENT SYSTEM PACKAGE INCLUDES:
• FULL-SCALE IMPLEMENTATION (K&R) C
COMPILER • THE STANDARD C LIBRARY • ROM
ROUTINES LIBRARY • LINKER • LIBRARIAN AND
DOCUMENTATION ...

\$299.95

FOR MORE INFORMATION OR TO ORDER CALL OR WRITE:
Megamax, Inc.
BOX 851521, DEPT. Y
RICHARDSON, TX
75085-1521
(214) 987-4931



MACINTOSH IS A
REGISTERED TRADEMARK
OF APPLE COMPUTER INC.

**NOW
AVAILABLE
FOR THE
MACINTOSH**

Circle no. 84 on reader service card.

Listing Two

```

fred    struc
dd      1 dup (?)      ;declare a structure macro
fred    ends           ;consisting of a double-word
                        ;data declaration

harry   fred    <>      ;invoke the structure macro

dseg    ends

cseg    segment para public 'CODE'

pgm     proc      far
ret     ;this is a Zen program...
pgm     endp

cseg    ends

        end      pgm

```

End Listing Two

Listing Three

General purpose Binary-to-ASCII conversion and unsigned divide subroutines.

```

;
; Convert 32 bit binary value to ASCII string.
; Copyright (C) 1984 Ray Duncan
;
; Call with  DX:AX = signed 32 bit value
;            CX    = radix
;            SI    = last byte of area to store resulting string
;                  (make sure enough room is available to store
;                  the string in the radix you have selected.)
;
; Destroys AX, BX, CX, DX, and SI.
;
bin_to_asc proc near      ;convert DX:AX to ASCII.
                        ;force storage of at least 1 digit.
        mov     byte ptr [si], '0'
        or      dx, dx    ;test sign of 32 bit value,
        pushf                    ;and save sign on stack.
        jns     bin1      ;jump if it was positive.
        not     dx        ;it was negative, take 2's complement
        not     ax        ;of the value.
        add     ax, 1
        adc     dx, 0

bin1:                                ;divide the 32 bit value by the radix
                        ;to extract the next digit for the
                        ;forming string.
        mov     bx, ax
        or      bx, dx
        jz      bin3      ;yes, we are done converting.
        call    divide    ;no, divide by radix.
        add     bl, '0'    ;convert the remainder to an ASCII digit.
        cmp     bl, '9'    ;we might be converting to hex ASCII,
        jle     bin2      ;jump if in range 0-9,
        add     bl, 'A'-'9'-1 ;correct it if in range A-F.
bin2:    mov     [si], bl    ;store this character into string.
        dec     si          ;back up through string,
        jmp     bin1        ;and do it again.
bin3:                                ;restore sign flag,
        popf                    ;was original value negative?
        jns     bin4      ;no, jump
        mov     byte ptr [si], '-' ;yes, store sign into output string.

```



```

bin4:    ret                      ;back to caller.
bin_to_asc endp
;
;
; General purpose 32 bit by 16 bit unsigned divide.
; Copyright (C) 1984 Ray Duncan
;
; This must be used instead of the plain machine unsigned divide
; for cases where the quotient may overflow 16 bits (for example,
; dividing 100,000 by 2). If called with a zero divisor, this
; routine returns the dividend unchanged and gives no warning.
;
; Call with DX:AX = 32 bit dividend
;          CX      = divisor
;
; Returns  DX:AX = quotient
;          BX      = remainder
;          CX      = divisor (unchanged)
;
divide proc near                ; Divide DX:AX by CX
    jcxz divl                   ; exit if divide by zero
    push ax                     ; 0:dividend_upper/divisor
    mov ax,dx
    xor dx,dx
    div cx
    mov bx,ax                   ; BX = quotient1
    pop ax                      ; remainder1:dividend_lower/divisor
    div cx
    xchg bx,dx                  ; DX:AX = quotient1:quotient2
    ret                          ; BX = remainder2
divl:
divide endp
;
;

```

End Listings

C Source Code

RED

Full Screen Text Editor

IBM PC, Kaypro, CP/M 80 and CP/M 68K systems.

- RED is fast! RED uses all of your terminal's special functions for best screen response. RED handles files as large as your disk automatically and quickly.
- RED is easy to use for writers or programmers. RED's commands are in plain English.
- RED comes with complete source code in standard C. RED has been ported to mainframes, minis and micros.
- RED comes with a Reference Card and a Reference Manual that provides everything you need to use RED immediately.
- RED is unconditionally guaranteed. If for any reason you are not satisfied with RED your money will be refunded promptly.

RED: \$95

Manual: \$10



Call or write today for more information:
Edward K. Ream
1850 Summit Avenue
Madison, WI 53705
(608) 231-2952

To order:

Either the BDS C compiler or the Aztec CII compiler is required for CP/M80 systems. Digital Research C compiler v1.1 is required for CP/M 68K systems. No compiler is required for IBM or Kaypro systems.

Specify both the machine desired (IBM, Kaypro or CP/M) and the disk format described (8 inch CP/M single density or exact type of 5 1/4 inch disk).

Send a check or money order for \$95 (\$105 U.S. for foreign orders). Sorry, I do NOT accept phone, credit card, or COD orders. Please do not send purchase orders unless a check is included. Your order will be mailed to you within one week.

Dealer inquiries invited.

The Tools You Need To C You Thru.

Now the *WizardWare*™ Applications Programmers Toolkit provides everything you need to increase your C programming productivity.

APT™ features include:

- COMPLETE SOURCE CODE (over 5000 lines!)
- File handling with direct & keyed access
- Screen and Report Generators, with full screen handling for your programs
- Generic Terminal Driver for portable code
- String math functions, and string manipulation routines
- Reference Manual on Disk (over 50 pages)
- Tutorial Manual (over 25 pages) with Source for Mailing List Manager
- A host of useful Utilities, Database and File Editors
- Available for Lattice C, Mark Williams C, DeSmet C, BDS C, others.

Also Available: C-STARTER Toolkit, great for learning C!! Includes: Customized APT, DeSmet C Compiler, and "Programming in C on the IBM-PC" (200 pages)

APT/MS-DOS versions	\$495
APT/DeSmet C version	\$395
APT/BDS C version	\$395
C-STARTER (binary APT, DeSmet Compiler and Book)	\$295
APT/Manual only	\$ 50

Detailed Brochures on request

*Manual Cost will be applied if APT purchased within 30 days (\$10 re-stocking charge.) U.S. funds only, please.

Trademark: MS DOS/Microsoft, Lattice C/Lattice, Inc., WPW/Mark Williams C, DeSmet C/C-Start, C/CBS/Computer Innovations, Inc., BDS C/BDS Software, DR C/Digital Research, WizardWare, APT, C-STARTER/Shaw - American Technologies

Call (502) 583-5527

Ask for APT™ or C-STARTER, or Send Check to:

Shaw ☆ American Technologies

WizardWare™

**830 South Second St. - Box 648
Louisville, KY 40201, USA**

(C.O.D. and Foreign Orders - Add \$5 Shipping/Handling)

References: Bank of Louisville, Citizens Fidelity Bank, Louisville Chamber of Commerce

Circle no. 86 on reader service card.

Announcing BOUND VOLUME 7

Every 1982 Issue Available For Your Personal Reference.

Vol. 7 1982

In 1982 we introduced several significant pieces of software, including the RED text editor and the Runic extensible compiler, and we continued to publish utility programs and useful algorithms. Two new columns, The CP/M Exchange and The 16-Bit Software Toolbox, were launched, and we devoted special issues to FORTH and telecommunications. Resident Intern Dave Cortesi supplied a year of "Clinic" columns while delivering his famous review of JRT Pascal and writing the first serious technical comparison of CP/M-86 and MSDOS. This was also the year we began looking forward to today's generation of microprocessors and operating systems, publishing software for the Motorola 68000 and the Zilog Z8000 as well as Unix code. And in December, we looked beyond, in the provocative essay, "Fifth-generation Computers."

Vol. 1 1976

The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering," front-panel, machine-language programming which was then the only way to do things. This is always pertinent for bit crunching and byte saving, language design theory, home-brew computer construction and the technical history of personal computing.

Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

Vol. 2 1977

1977 found DDJ still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 8080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic, and NIBL. Articles are about Lawrence Livermore Lab's BASIC, Alpha-Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

Vol. 3 1978

The microcomputer industry entered its adolescence in 1978. This volume

brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this, they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today. Languages covered in depth were SAM76, Pilot, Pascal, and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors, and much, much more.

Vol. 4 1979

This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages. Innovation is still present in every page, and more attention is paid to the best ways to use the processors which have proven longevity—primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.

Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit Conversion, Pseudo-random Sequences, and Interfacing a Micro to a Mainframe—more than ever!

Vol. 5 1980

All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (DDJ's all-CP/M issue sold out within weeks of publication). Software portability became a topic of greater import, and DDJ published Ron Cain's immediately famous Small-C compiler—reprinted here in full!

Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080, Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Run-time Library for the Small-C Compiler and, as always, even more!

Vol. 6 1981

1981 saw our first all-FORTH issue (now sold out), along with continuing coverage of CP/M, small-C, telecommunications, and new languages. Dave Cortesi opened "Dr. Dobb's Clinic" in 1981, beginning one of the magazine's most popular features. Highlights: information on PCNET, the Conference Tree, and The Electric Phone Book, writing your own compiler, a systems programming language, and Tiny BASIC for the 6809.

YES!

- ☐ Please send me the following Volumes of Dr. Dobb's Journal.
☐ ALL 7 for ONLY \$165, a savings of over 15%!

Payment must accompany your order.

Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order

Card # _____ Expiration Date _____

Signature _____

Name _____ Address _____

City _____ State _____ Zip _____

Mail to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303

Vol. 1	x	\$26.75	=	_____
Vol. 2	x	\$27.75	=	_____
Vol. 3	x	\$27.75	=	_____
Vol. 4	x	\$27.75	=	_____
Vol. 5	x	\$27.75	=	_____
Vol. 6	x	\$27.75	=	_____
Vol. 7	x	\$30.75	=	_____
All 7	x	\$165.00	=	_____

Sub-total \$ _____

Postage _____

Postage & Handling Must be included with order.

Please add \$1.25 per book in U.S. (\$3.25 each surface mail outside U.S.) Foreign Airmail rates available on request.

TOTAL \$ _____

DR DOBB'S JOURNAL BRINGS RESTON BOOKS TO YOU

SCIENTIFIC PASCAL

by Harley Flanders

A refreshing, stimulating book on the most rapidly growing structured programming language. The book is filled with over 400 examples and exercises with graded levels of difficulty. The state-of-the-art programs involve approximation, zeros of functions, differentiation and integration, differential equations, matrices, characteristic roots, fast Fourier transform, permutations—anything and everything connected with scientific programming and numerical analysis.

1984 576 pages \$21.95 (soft)

THE PROGRAMMER'S CP/M NOTEBOOK

by David E. Cortesi

Displays the process of designing and building software for the popular CP/M operating system. Twelve complete, useful utility programs are contained in the book. For each program the author analyzes the user's needs and develops a verbal specification, then designs the program in a high-level language, and presents the assembly language code that implements the program.

1983 225 pages \$17.95 (soft)

PARALLEL PROGRAMMING IN ANSI STANDARD ADA

by George Cherry

A thorough introduction to parallel programming and algorithms in the military and civilian standard language, Ada. Uses graphical means (Petri nets) to introduce the basic

concepts of concurrency, synchronization, and rendezvous. Illustrates the language features with complete programs rather than program fragments. Treats task pipelines, exception handling in parallel programs, and algorithms that are optimized for the number of processors available.

1984 224 pages \$21.95 (hard)

SYSTEM PROGRAMMING UNDER CP/M-80

by Lawrence E. Hughes

A thorough introduction to 8080/8085 assembly language programming. Contains a great deal of data on how to program under the CP/M operating system. Gives detailed information on installing and modifying CP/M. Includes actual usable utility programs for CP/M, along with a detailed analysis of each.

1983 208 pages \$17.95 (soft)

A USER FRIENDLY GUIDE TO CP/M

by James T. Perry and Robert F. McJunkins

This guide is perfect for first-time users—taking them from an introduction to CP/M, to creating new dBASE commands. In between, the reader will find topics such as built-in commands, copying a large number of files, system files, user area, copying selected parts of files, introduction to dBASEII, altering the data base contents, and more.

1983 150 pages \$16.95 (soft)

PERSONAL PASCAL: COMPILED PASCAL FOR THE IBM PERSONAL COMPUTER

by David E. Cortesi and George W. Cherry

Prepares the reader to use the Pascal programming language and compiled Pascal on the IBM Personal Computer in particular. Illustrates stepwise refinement to help the reader complete designs. Gives careful descriptions and illustrations of Pascal syntax to aid the reader in avoiding common syntactic errors. Emphasizes clarity of design and protability of code throughout.

1984 432 pages \$17.95 (soft)

THE SMALL-C HANDBOOK

by James E. Hendrix

A complete description of the Small-C programming language and compiler for those who are already programming in other languages. Section 1 covers program translation concepts, presenting a survey of machine language concepts, assembly language, and the use of assembler loaders and linkers. Section 2 introduces the Small-C language, and Section 3 describes the compiler itself, covering I/O concepts, standard functions, using the compiler to generate new versions of itself, and more. In addition, several appendices offer valuable reference information on Small-C programming.

1984 272 pages \$17.95 (soft)



Mail coupon to: Dr. Dobb's Journal/2464 Embarcadero Way/Palo Alto, CA 94303

Yes! Please send me the book(s) in the quantities I've indicated below:

- Scientific Pascal, soft, R6931-1, \$21.95
- The Programmer's CP/M Notebook, soft, R5641-7, \$17.95
- Parallel Programming in ANSI Standard ADA, hard, R5434-7, \$21.95
- The Small-C Handbook, soft, R7012-9, \$17.95
- Personal Pascal: Compiled Pascal for the IBM Personal Computer, soft, R5522-9, \$17.95
- System Programming Under CP/M-80, soft, R7456-8, \$17.95
- A User-Friendly Guide to CP/M, soft, R8117-5, \$16.95

☐ Enclosed is my payment of _____ in check or money order.

☐ VISA ☐ MasterCard

Card No.: _____

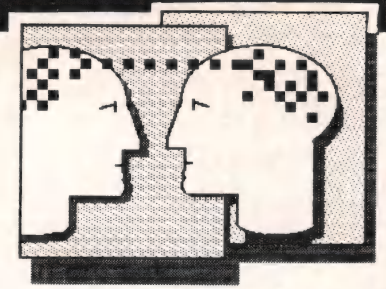
Expiration date: _____

Name: _____

Address: _____

City: _____ State: _____ Zip: _____

Please allow 4-6 weeks for delivery. Add \$1.50 per book for shipping and handling charges.



by Robert A. Blum

Steve Russell of SLR Systems, the author of Z80ASM, recently sent along a disk containing a problem report that he had sent to DRI. The disk contained a program to demonstrate the problem he had found with CP/M Plus and the RSX that he had written to correct the problem.

While installing for a construction company an accounting package that was written in a high-level language, Steve encountered inconsistent results when he attempted to determine the current disk file sizes. Although the software publisher proved to be of little help, Steve did learn that this strange occurrence seemed also to appear under MP/M but never with CP/M 2.2—and that the disk files in question were being accessed with random I/O BDOS calls.

After a great deal of probing and prodding into how CP/M Plus was responding to the application program's system calls, Steve found a problem with the way the CP/M Plus file system was treating random read errors. This situation was compounded further by the fact that errors sometimes were not reported at all and that incorrect data was being placed into the FCB.

The disk I received contained a sample data file, 257 sectors (records) in length, and a program FIND that would produce the problem. When run, the FIND program showed that the random record logic of CP/M Plus became confused on the second consecutive random read to an unallocated sector of an existing extent other than extent 0. However, if an interceding valid read is done or a read to an unallocated record of an unwritten extent is done, the problem does not occur.

The FIND program (Listing One, page 112) tries to read several sectors from the file XXX. Each read attempt is documented by displaying the record number in hex and the return error

code in A. The FCB contents then are dumped to find out what is happening.

Follow along in Figure 1 (page 109) as each read is documented. The first read is to record zero, which is an allocated record of extent zero; it correctly completes with a zero return code and valid FCB contents. The second read is to record number 257, which is one record beyond the last record written to the file. The return code for this read is reported correctly as one, indicating that the last read operation was on an unallocated record of a written extent. The third read is to an allocated record of extent zero, providing once again a valid error return code of zero and correct FCB contents.

The error occurs on the second of the next two reads. The fourth read is to record number 500, an unallocated record of a written extent. The read is flagged correctly as an error, and the FCB contents are also correct. Now, the error occurs. The fifth read is to record number 260, which again is to an unallocated record of a written extent. This time, the error return code is reported incorrectly as zero, or no error, and the record count field of the FCB incorrectly contains 80h. In reality, both the return code and the record count field should have been one.

So, the problem seems to happen only when two consecutive random reads to an unallocated record of a written extent are attempted. To correct the problem, Steve wrote an RSX (Listing Two, page 114) to monitor the BDOS function calls made by the application program. When a random read is found, the return error code is examined, and when a nonzero error return code is found, a dummy read to record zero of the same file is done. This interceding read keeps the incorrect error return code and FCB contents from resulting, should the same type of read occur a second consecutive

time. You might want to examine Figure 2 (page 110), which displays the results of the FIND program after the installation of the RSX.

CP/M Plus: PIP Patch

Gary Silvey of Digital Research recently sent along a patch for the CP/M Plus version of PIP to set the verify option permanently on, as I described for CP/M 2.2 in the November issue of this column. As before, DRI does not recognize this patch, even though it comes from a DRI employee. Also, the patch is known to work only with the specific version of the PIP program mentioned here. Before setting out to install this patch, make a backup copy of your disk in case it becomes necessary to return to your original version of PIP.

Figure 3 (page 112) lists the procedure that I followed to install the verify patch on my system. The first step is to verify that your PIP program is version 3.0; this is the only version of PIP with which this patch is known to work successfully. To verify your version number, load PIP.COM into memory with either SID or DDT and display memory between 210h and 21fh; the version number, which should be 3.0, should begin at 21Ch. The next step is to change the single byte at 1001h from 1Fh to 37h. The final step is to save the modified program back onto disk under a new name ready for testing.

To ensure that I had entered my patches correctly, I timed how long it took to copy the same file with an unaltered version of PIP using the verify option and again with the newly altered version of PIP. The two times were practically identical, which was verification enough for me.

Gary also sent along the timings from his testing of the changes; these once again point out how much quicker

disk I/O is under CP/M Plus.

For your information, I copied HELPHLP to a file called H to check out the PIP version 3.0 patch. Under my nonbanked version of CP/M Plus, it took 13:42 sec for the copy to complete

with the patched PIP and 21:71 sec with unpatched PIP. On my banked system, copying the same file with the patched and unpatched PIP yielded times of 8:00 and 18:24 sec, respectively. Using the unpatched version of PIP

on the banked system with the V option yielded 18:25 sec. DDJ

(Listing begins on page 112)

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service No. 196.

```
READING RECORD 0000, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 00 00 00
```

```
READING RECORD 0101, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01 01 01 00
```

```
READING RECORD 0064, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
64 64 00 00
```

```
READING RECORD 01F4, ERROR CODE RETURNED IN A IS 01.
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 03 02 80 00
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
74 F4 01 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 80
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04 04 01 00
```

```
READING RECORD 0000, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 00 00 00
```

```
READING RECORD 0100, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04 04 01 00
```

```
READING RECORD 0202, ERROR CODE RETURNED IN A IS 04
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
02 02 02 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04 04 01 00
```

Figure 1 Results of Random Read Before RSX Installed.

PROGRAM PACKAGING DISKS, SUPPLIES

- IBM style cloth binders and slips for 5½ x 8½. Imprinting available in small quantities.
- Vinyl binders for 5½ x 8½ and 8½ x 11 with clear pockets on front and spine for labeling.
- Vinyl packaging in several styles.
- Floppy disk pages for 5¼ and 3½ disks.
- Continuous paper for IBM size binders with 3 holes punched. 2500 sheets for only \$40.00.
- 5¼ DS/DD bulk disks—\$149/100. No labels or envelopes. Blank & printed envelopes & labels available.
- Low prices on packaged disks—BASF, KODAK, Elephant, Memorex. Disks for IBM AT and Mac in stock.

• Many other products. Call or write for FREE CATALOG. Fast service.

Anthropomorphic Systems, Limited

Department DDJ

376 E. St. Charles Road

Lombard, IL 60148

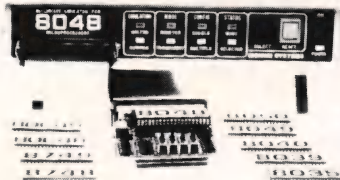
(312) 629-5160



Circle no. 11 on reader service card.

IN-CIRCUIT EMULATOR

For 8050, 8049 and 8048 µC's



For Hardware/Software Design the E232-48 replaces the target microcomputer for complete emulation under control of the host computer. It emulates all of the above µC's plus their ROM-less and CMOS versions. It's features are:

- ★ Real time emulation up to 11 Mhz.
- ★ Full 4K Emulation Memory.
- ★ Hardware Breakpoints.
- ★ In-line Assembler and Disassembler.
- ★ Upload, Download & Terminal mode drivers for IBM-PC, CP/M-80 and CP/M-86 are included.

Cross Assemblers for 8048 series and other µP's running on the above host systems—From \$150
E232-48 in-circuit emulator..... \$1795

SIGNUM SYSTEMS

726 Santa Monica Blvd.
Santa Monica, CA 90401 (213) 451-5382

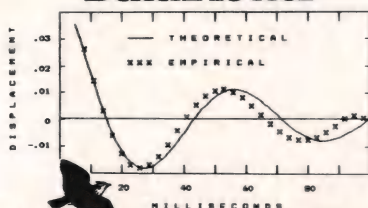
Circle no. 106 on reader service card.

Graphs without Graphics?

No need for screen graphics. Publishable graphs on your dot matrix printer.

Easy to Use. No programming.
CP/M 80 or 86, MS-DOS, or PC-DOS.
Excellent Manual. Most disk formats.

DataPlotter™



Lark Software™
7 Cedars Road
Caldwell, NJ 07006

Line Graphs &
Scatterplots...\$69
Bar Graphs &
Pie Charts...\$69
Both for ...\$99
(Prices include manual)

Shipping..... add \$3
Outside US and
Canada..... add \$6
Specify which Printer
Visa, M/C
(201) 226-7552

Circle no. 57 on reader service card.

```
READING RECORD 0000, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 00 00 00
```

```
READING RECORD 0101, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 01 01 00
```

```
READING RECORD 0064, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
64 64 00 00
```

```
READING RECORD 01F4, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 F4 01 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 04 01 00
```

```
READING RECORD 0000, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 00 00 00
```

```
READING RECORD 0100, ERROR CODE RETURNED IN A IS 00
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 02 02 80 01
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 01 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 04 01 00
```

```
READING RECORD 0202, ERROR CODE RETURNED IN A IS 04
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
02 02 02 00
```

```
READING RECORD 0104, ERROR CODE RETURNED IN A IS 01
36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ
00 58 58 58 20 20 20 20 20 20 20 20 00 02 80 80
06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
00 04 01 00
```

Figure 2

Results of Random Read After RSX Installed.

Powerful in circuit emulation, priced well within your grasp. That's NICE.™

NICE may be only 3" square and 1/2" thick, but it hands you full speed, real-time emulation—over 50 emulation functions, software breakpoints, all memory addresses and all I/O ports.

Just plug NICE directly into the target MP socket and any RS232 terminal for system development, troubleshooting, debugging or testing... at home, in the lab or in the field.

And NICE hands you all this performance, portability and versatility for only \$498*... the best emulator price/performance ratio on the market, hands down.

Call in your order today using

your VISA or MasterCard number: (800) NICOLET outside CA, or (415) 490-8300 in CA.

Or send your check or money order to NICE, Nicolet

Paratronics Corporation, 201 Fourier Avenue, Fremont, CA 94539.

*Payment by check, money order, VISA or MasterCard

NICE is a trademark of Nicolet Paratronics Corporation

*Z80 is a trademark of Zilog, Inc.

Nicolet

NOW AVAILABLE—NICE for the Z80
COMING: NICE 8085—Sept. '84
NICE NSC800—Oct. '84
MORE TO COME!!

Circle no. 60 on reader service card.

\$5.00 C Compiler

Due to popular demand, **Dr. Dobb's Journal** has reprinted its most-asked-for C compiler articles by Ron Cain and J. E. Hendrix, each for only \$5.00.

Ron Cain's C compiler from sold-out 1980 issues #45 and #48 includes "A Small C Compiler for the 8080s" and "Runtime Library for the Small C Compiler."

The J. E. Hendrix reprint includes part two of "Small-C Compiler v.2" from sold out issue #75 and completes the first part of the compiler article from issue #74 which is included in Dr. Dobb's Bound Volume 7.

To Order: Enclose \$5.00 for each copy with this coupon and send to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303 Outside U.S., add \$2.00 per copy for shipping and handling.

Please send _____ copy(ies) of the Ron Cain Reprint, and
_____ copy(ies) of the J. E. Hendrix reprint to:

Name _____

Address _____

City _____ State _____ Zip _____

ALL REPRINT ORDERS MUST BE PREPAID.

101

RUN/C:™ Finally, a C Interpreter

Available NOW for only \$149.95!

Finally, a painless introduction to the C language. With **RUN/C: The C Interpreter** you can create and run C language programs in an environment as easy to use as BASIC.

RUN/C is C for the rest of us. It is a robust implementation of standard K&R. **RUN/C** is for both the beginner and professional.

RUN/C includes full floating point, 8087 support, structures, unions, casts and more than 100 built-in C functions.

With **RUN/C** you get all this with a command structure modeled after BASIC's using familiar terms such as EDIT, RUN, LIST, LOAD, SAVE, TRON, SYSTEM, etc.

Since **RUN/C** is a true interpreter it means that C programs can be written, tested and run within a single protected environment. It is a teaching tool and a source code debugger.

Here's more good news...

- Great documentation: a 400-page, easy-to-read manual filled with executable programs
- Array-index and pointer bounds checking
- Variable-trace and dump diagnostics PLUS an integral program profiler
- Full buffered and unbuffered file I/O
- Printer and asynch support
- Forking to your favorite full screen editor with automatic return to **RUN/C** with your edited program
- System Requirements: IBM® PC or compatible with PC-DOS 2.0 or MS™-DOS 2.0 or greater with ANSI.SYS.

Get things right the first time with **RUN/C: The C Interpreter.™**

For immediate delivery or more information, call:

1-800-847-7078

(in N.Y. 1-212-860-0300)

or write: Lifeboat Associates™
1651 Third Avenue
New York, NY 10128

RUN/C is a trademark of Age of Reason Co.

Circle no. 74 on reader service card.


```

B>SID PIP.COM
CP/M 3 SID - VERSION 3.0
NEXT MSZE PC    END
2300 2300 0100 C8FF
#D210,21F
0210  20 33 20 50 49 50 20 56 45 52 53 20 33 2E 30 20  3 PIP VERS 3.0
#S1001
1001 1F 37
1002 D2 .
#WPIPNEW.COM,100,2300
0044h record(s) written.
#GO

```

Figure 3
Pip verify patch

CP/M Exchange (Text begins on page 108)

Listing One

```

TITLE    DEMONSTRATE FAULTY RANDOM RECORD ERROR
DFCB     EQU    5CH

;
; THE SAMPLE FILE NAME IS RATHER GENERIC, 'XXX'
; TO RUN THIS PROGRAM, JUST TYPE A>FIND XXX
;
START
;
;FIRST OPEN THE FILE
;
        LD      DE,DFCB
        LD      C,0FH
        CALL    5
;
;DID IT OPEN ?
;
        INC     A
        RET     Z           ;NOPE, RETURN
;
;OK, MAKE SURE HIGH BYTE OF RANDOM RECORD IS 0
;
        XOR     A
        LD      (DFCB+35),A
;
        LD      SP,MYSTACK
;
;FIRST, LETS SET UP A STRING OF RECORD NUMBERS TO READ
;WITH A RANDOM READ TO SEE IF THEY REACT DIFFERENTLY
;
        LD      HL,RECORD_TABLE           ;TABLE OF RECORD NUMBERS
        LD      A,TABLE_LEN              ;LENGTH OF TABLE
LOOP1:
        PUSH    AF                       ;SAVE COUNT
        LD      E,(HL)                   ;LOAD RECORD # DESIRED
        INC     HL
        LD      D,(HL)
        INC     HL
        PUSH    HL                       ;SAVE TABLE POINTER
        EX      DE,HL
        LD      DE,RECNUM                 ;ENCODE RECORD NUMBER TO HEX-ASCII
        CALL    HEXWOUT
        LD      (DFCB+33),HL              ;SET RECORD NUMBER
        LD      DE,DFCB
        LD      C,21H                     ;RANDOM READ
        CALL    5
        LD      DE,RESULT                 ;ENCODE VALUE RETURNED IN A
        CALL    HEXOUT
        CALL    PRINT_LINE                ;PRINT LINE ON THE CONSOLE
        POP     HL                        ;RESTORE POINTER AND COUNTER
        POP     AF

```



```

        DEC     A
        JR      NZ,LOOP1      ;IF MORE RECORDS, JUMP
;
;FINISHED.....
;
        JP      0

RECORD_TABLE
        DW      0              ;FIRST READ RECORD ZERO
        DW      257            ;READ RECORD 257 (ONLY 0-256 EXIST...ERROR REPORTED
        DW      100            ;READ RECORD 100 (MAIN PURPOSE, BACK TO EXT 0)
        DW      500            ;READ RECORD 500 (ONLY 0-256 EXIST...ERROR REPORTED
        DW      260            ;READ RECORD 260 (ONLY 0-256 EXIST...OOPS! NO

R!)
        DW      0              ;BACK TO VALID EXT INFO
        DW      256            ;LAST RECORD OF FILE, FOUND
        DW      260            ;READ 260, ERROR RETURNED
        DW      514            ;READ RECORD 514 ( " " " ,NOTE, UNWRITTEN EXT
        DW      260            ;READ RECORD 260 ( " " " ,THIS TIME ERROR

REPORTED)
TABLE_LEN EQU      ($-RECORD_TABLE)/2

HEXWOUT
        LD      A,H
        CALL    HEXOUT
        LD      A,L

HEXOUT
        PUSH    AF
        RRA
        RRA
        RRA
        RRA
        CALL    HEX2
        POP     AF

HEX2
        AND     0FH
        OR      30H
        CP      3AH
        JR      C,HEX3
        ADD     7

HEX3
        LD      (DE),A
        INC     DE
        RET

PRINT_LINE
        LD      DE,RECLINE
        LD      C,9
        CALL    5

;
;NOW PRINT OUT THE FCB
;
        LD      HL,DFCB
        LD      DE,FCBASCII
        LD      B,36

LOOP5
        LD      A,(HL)
        INC     HL
        CALL    HEXOUT
        INC     DE
        DJNZ    LOOP5
        LD      DE,FCBASC
        LD      C,9
        JP      5

FCBASC DB      0DH,0AH
        DB      '36 HEX BYTES OF FCB CONTENTS AFTER ATTEMPTED READ',0DH,0AH
FCBASCII
        DS      36*3,20H
        DB      0DH,0AH,'$'

RECLINE DB      0DH,0AH,'READING RECORD '
RECNUM  DB      '      , ERROR CODE RETURNED IN A IS '
RESULT  DB      '  $ '

        DS      64

MYSTACK

        END

```

End Listing One

(Listing two begins on next page)


```

        TITLE  RANDOM READ RSX SOURCE LISTING
;START OF PREFIX
DB      0,0,0,0,0,0      ;SERIAL NUMBER
JP      START
NEXT    JP      $-$
PREV    DW      0
REMOVE  DB      0
NONBNK  DB      0
RSXNAME DB      'TRACE EM'
        DB      0,0,0
;
;START OF RSX CODE
;
START
        LD      A,C          ;PUT BDOS FUNCTION CODE IN A
        CP      21H          ;IS IT A RANDOM READ
        JR      NZ,NEXT      ;SKIP IF NOT RANDOM READ
START2
;
;      DO READ, BUT IF IT RETURNS AN ERROR CODE 1
;      SEEK SECTOR 0 BEFORE GOING ON TO CORRECT POSSIBLE
;      INACCURACY OF SUBSEQUENT RANDOM READ
;
        LD      (USERSAVE),SP ;SAVE USER STACK POINTER
        LD      SP,MYSTACK    ;LOAD A LOCAL STACK POINTER
        PUSH    DE            ;SAVE FCB ADDRESS
        CALL    NEXT          ;ALLOW REQUESTED OPERATION TO COMPLETE
;
;      THE USER REQUESTED OPERATION IS NOW COMPLETE
;
        CP      1             ;WAS THE RETURN CODE DATA UNAVAILABLE
        JR      NZ,START8     ;NO - BYPASS OUR FIX
;
;      SAVE ALL THE USER INFORMATION AND PREPARE FOR SEEK TO SECTOR ZERO
;
        EX      (SP),HL        ;SAVE HL AND LOAD FCB ADDRESS
        LD      DE,33          ;GET INDEX TO R0 POSITION OF FCB
        EX      DE,HL
        ADD     HL,DE          ;ADD R0 POSITION INDEX TO FCB ADDRESS
        PUSH    HL             ;SAVE ON STACK
        XOR     A              ;PUT ZERO INTO A
        LD      C,(HL)         ;PUT FCB R0 BYTE INTO C
        LD      (HL),A         ;ZERO FCB R0 BYTE
        INC     HL             ;POINT AT R1 BYTE
        LD      B,(HL)         ;PUT FCB R1 BYTE INTO B
        PUSH    BC             ;SAVE BC ON STACK
        LD      (HL),A         ;ZERO FCB R1 BYTE
        INC     HL             ;POINT AT R2 BYTE
        LD      B,(HL)         ;PUT FCB R2 BYTE INTO B
        PUSH    BC             ;SAVE BC ON STACK
        LD      (HL),A         ;ZERO FCB R2 BYTE
        LD      C,21H          ;GET RANDOM READ FUNCTION CODE
        CALL    NEXT           ;EXECUTE IT
;
;      NOW RESTORE ALL THE ORIGINAL USER INFORMATION
;
        POP     AF             ;PULL R0, R1, AND R2 INFO FROM THE STACK
        POP     BC             ;*
        POP     HL             ;PULL THE FCB ADDRESS FROM THE STACK
        LD      (HL),C         ;PUT THE ORIGINAL RANDOM READ INFO BACK
        INC     HL             ;INTO THE FCB
        LD      (HL),B         ;*
        INC     HL             ;*
        LD      (HL),A         ;*
        LD      A,1            ;RESTORE THE ERROR CODE
        POP     HL             ;CORRECT STACK
START8
        LD      SP,(USERSAVE)  ;RESTORE USER STACK POINTER
        RET                     ;AND BACK WE GO
USERSAVE DW      0
        DS      64
MYSTACK
END

```

End Listings

FOR THE BEST OF US...
THE CYPHER™
 A COMPLETE 68000 & Z 80
 SINGLE BOARD COMPUTER SYSTEM
 WITH ULTRA-HIGH-RES GRAPHICS!!

NEW REDUCED PRICES!



NEW REDUCED PRICES!

- 68000 & Z80 DUAL PROCESSORS (BEST OF BOTH WORLDS)
- 256K TO 1 MEGABYTE MEMORY (415A OR 4125A DRAM)
- DOUBLE DENSITY FLOPPY DISK CONTROLLER (8" OR 5.25" HD 2HD)
- DMA CONTROLLER FOR FAST IMAGE TRANSFERS TO PROM VIDEO MEMORY (INT 8237)
- 2 RS232 SERIAL PORTS (2500)
- 24 BIT ADDRESS MANAGEMENT FOR Z80
- 4 LAYER PCB (9" x 14")
- RUNS CP/M-80 AND CP/M-86
- ULTRA-HIGH RESOLUTION GRAPHICS (128K PROGRAMMABLE UP TO 1024 X 1024 RESOLUTION (REC 1220 GREAT FOR CAD SYSTEMS))
- REAL TIME CLOCK (MULTITASKING CAPABILITY)
- TWO CHANNELS OF D/A AND 12 BIT RESOLUTION (MUSICO ROBOTICS)
- 16K TO 64K MONITOR EPROM
- 8K TO 64K STATIC RAM
- PROGRAMMABLE BAUD RATE GENERATOR
- PARALLEL ASCII KEYBOARD INPUT
- FULL 68000 EXPANSION BUS (50 PIN HEADER BUFFERED)

(80286 CPU PLUG-IN CARD AVAILABLE SOON)

MANUAL	\$ 20.00	CONTROLLER SERIAL I/O ASSEMBLED AND TESTED	\$1,499.95
• BASIC BOARD	\$ 280.00	80286 CPU PLUG-IN CARD	\$1,799.95
• MONITOR 68000 MONITOR AND UTILITIES	\$ 399.95	ASSEMBLED AND TESTED	\$ 154.95
• MINIMUM OPTIONED: CYPHER WITH 68000/250 SERIAL	\$ 1,099.95	ASSEMBLED AND TESTED	\$ 154.95
• 128K DRAM 4K 5 RAM AND DISK CONTROLLER	\$ 1,099.95	ASSEMBLED AND TESTED	\$ 154.95
• COMPLETE CYPHER WITH 256K DRAM 128K VIDEO	\$ 999.95	ASSEMBLED AND TESTED	\$ 154.95
• COMPLETE CYPHER WITH 256K DRAM 128K VIDEO	\$ 999.95	ASSEMBLED AND TESTED	\$ 154.95

ALL PRICES ARE IN U.S. DOLLARS. SHIPPING AND HANDLING CHARGES WILL BE ADDED TO ALL ORDERS. DELIVERY TIME: 4-6 WEEKS. PAYMENT: CASH, CREDIT CARD, OR CHECK. WE CAN SHIP VIA FED-EX ON SAT. (416) 221-2340

Circle no. 56 on reader service card.

ICs PROMPT DELIVERY!!!
 SAME DAY SHIPPING (USUALLY)

OUTSIDE OKLAHOMA: NO SALES TAX

8087-3	Co-Processors	\$131.25
DYNAMIC RAM		
256K	256Kx1	150 ns \$12.79
128K	128Kx1	150 ns 17.50
64K	64Kx1	150 ns 2.44
64K	64Kx1	200 ns 2.57
EPROM		
27256	32Kx8	300 ns \$34.97
27128	16Kx8	250 ns 10.97
27C64	8Kx8	200 ns 11.87
2764	8Kx8	250 ns 5.31
2732A	4Kx8	250 ns 5.99
2716	2Kx8	450 ns 3.21
STATIC RAM		
6264P-15	8Kx8	150 ns \$18.75
6116P-3	2Kx8	150 ns 3.75

QUANTITY ONE PRICES SHOWN

OPEN 6 1/2 DAYS: WE CAN SHIP VIA FED-EX ON SAT.

MasterCard/VISA or UPS CASH COD
 Factory New, Prime Parts

MICROPROCESSORS UNLIMITED
 24,000 South Peoria Ave. (918) 267-4961
 BEGGS, OK 74421

Prices shown above are for January 21, 1985

Please call for current prices. Prices subject to change. Please expect higher or lower prices on some parts due to supply & demand and our changing costs. Shipping & insurance extra. Cash discount prices shown. Small orders received by 6 PM CST can usually be delivered to you by the next morning. Via Federal Express Standard Air @ \$6.75!

Circle no. 64 on reader service card.

No source code for
 your REL files?
REL/MAC

converts a REL file in the Microsoft™ M80 format to a ZILOG™ or 8080 source code MAC file with insertion of all public and external symbols.

- REL/MOD lists library modules
- REL/VUE displays the bit stream
- 50 page manual with examples
- free brochure available
- REL/PAK includes all of the above

REL/PAK for 8080 only \$99.95
 REL/PAK for Z80 & 8080 \$134.95
 on 8"SSD disk for CP/M™ 2.2

Send check, VISA, MC or C.O.D. to

MICROSMITH
 COMPUTER TECHNOLOGY
 P.O. BOX 1473 ELKHART, IN 46515
 1-800-622-4070
 (Illinois only 1-800-942-7317)

Circle no. 41 on reader service card.

cVIEW™ SCREEN MANAGER

cVIEW helps to create and communicate with forms that will enable you to provide a sophisticated user interface for your C applications programs.

**CI86, Lattice, Mark Williams
 Dynamic Forms Position
 Special Key Definitions
 Input Range Validation
 Color or Monochrome
 Block Cursor Option
 Selection Sets**

cVIEW Screen Manager	\$245.00
cVIEW with Source Code	\$545.00
cVIEW Demo	\$25.00

CompuCraft (313)
 42101 Mound Road 731-2780
 Sterling Heights, Michigan 48078

Circle no. 9 on reader service card.

ZAS
 Software
 Development Package

The industry's most sophisticated development tool is now available for the **Z-8000** and the **Z-8** under **CP/M, MDOS, and ISIS.**

Includes:

- ZAS Relocatable Macro Cross Assembler
- 38 directives, nested macros, etc.
- ZLK Task Builder/Linker
- resolves CALR's, section oriented
- ZLD User-Modifiable Object Loader
- ZEX Dual Processor Run-Time Support

CP/M™ Digital Research
 MDOS™ Microsoft
 ISIS™ Intel Corp.

Western Wares
 303-327-4898
 Box C • Norwood, CO 81423

Circle no. 118 on reader service card.

BYSO™ LISP

has features that will delight both beginners and advanced programmers. A fast, reliable and complete interpreter for the IBM PC and true compatibles. \$125 includes 100 pg. ref. manual and application notes that put you months ahead on useful projects (making a hybrid language with C, accessing system functions and I/O ports, building your own dialect, etc.).

LEVIEIN INSTRUMENT CO.
 P.O. Box 31B
 McDowell, VA 24458
 703-396-3345

IBM PC is a trademark of the IBM Corp.

Circle no. 25 on reader service card.

A general purpose programming language for string and list processing and all forms of non-numerical computation.

SNOBOL4+ — the entire SNOBOL4 language with its superb pattern-matching facilities • Strings over 32,000 bytes in length • Integer and floating point using 8087 or supplied emulator • ASCII, binary, sequential, and random access I/O • Assembly Language interface • Compile new code during program execution • Create SAVE files • Program and data space up to 300K bytes RAM

Have you tried SNOBOL4+?

With **ELIZA** & over 100 sample programs and functions

For all 8086/88 PC/MS-DOS or CP/M-86 systems, 128K minimum 5 1/4" DSDD, specify DOS/CPM format

Send check, VISA, M.C. to: **\$95**
Catspaw, Inc. plus \$3 s/h
 P.O. Box 1123 • Salida, CO 81201 • 303-539-3884

Circle no. 20 on reader service card.

**C Programmers
 B-Trees**

For **\$75.00** + 2.00 Postage
 Source Code Included

The Softfocus B-Trees record indexing library will help you develop sophisticated application programs. With Softfocus B-Trees you get:

- high speed file handling for up to 16.7 million records per file
- customizable BDS or K & R standard C source code
- no royalties on applications programs
- support random and sequential file access
- includes example programs

softfocus
 1277 Pallatine Dr., Oakville, Ont. Canada L6H 1Z1 (416) 844-2610

Circle no. 89 on reader service card.

Now With Windowing!
\$49.95 Basic Compiler

MTBASIC

Features:

Multitasking	Windowing
Handles interrupts	Interactive
Fast native code	Compiles quickly
Floating point	No runtime fee

MTBASIC is a true native code compiler. It runs Bytes's Sept. '81 seive in 26 seconds; interpreters take over 1400 seconds! Because MTBASIC is multitasking, it can run up to 10 Basic routines at the same time, while displaying ten separate windows. Pop-up/down menus are a snap to implement.

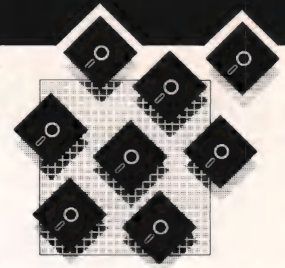
MTBASIC combines the best of interpreters and compilers. To the programmer, MTBASIC appears to be an extremely fast interpreter. MTBASIC compiles a typical 100 line Basic program in 1 second, yet it generates blindingly fast code. No more waiting for long compilations.

AVAILABLE for CP/M (Z-80) and PC-DOS systems.

ORDERING: Specify format when ordering. We accept Visa, MC, checks and COD. Send \$49.95 plus \$3.50 shipping and handling (\$10 overseas) to:

SOFTAID, Inc.
 P.O. Box 2412 Columbia, MD 21045-1412
 301/792-8096

Circle no. 88 on reader service card.



SAVVY PC, Version 4.0

Company: Excalibur Technologies Corporation, 800 Rio Grande Blvd. NW, Albuquerque, NM 87104, (505) 242-3333

Computer: IBM PC and most compatibles

Price: \$395

Circle Reader Service No. 131

Reviewed by Steve King

SAVVY PC is misrepresented—Excalibur Technologies advertises its product as “the artificial intelligence data base,” but the data base manager constitutes only a small part of the total contents of the SAVVY PC package. SAVVY is actually a complete programming language unto itself. Excalibur Technologies seems to have included the SAVVY PC data base manager more as an example of the SAVVY programming language’s capabilities. Don’t get me wrong; the SAVVY PC data base manager is as powerful as any I’ve tried, including KnowledgeMan. But the SAVVY programming language is the heart of the whole SAVVY PC package.

Fourth and Fifth Generation Languages

The mainframe world is undergoing a computer language revolution that partially parallels a similar change in the mini- and microcomputer worlds. Fourth generation languages are replacing the mainframe world’s liturgical high-level programming language, COBOL.

From a microcomputer world view, it appears anachronistic that a basically unstructured language, such as COBOL, continues to enjoy extensive use for applications programming. (Of course, the microcomputer world also continues to suffer with Microsoft BASIC in its many flavors.) But the existence of so much mainframe COBOL

code makes it economically unfeasible to discard the language in favor of such structured languages as Pascal or C, which have become so popular for mini- and microcomputers. However, fourth generation languages (FGLs) are now helping to solve this mainframe problem: they allow programmers to develop application programs much more rapidly than high-level languages, and moreover, many mainframe FGLs can access COBOL data files.

What is an FGL? It’s the procedural and query language supplied with a data base management system (DBMS); examples are dBASE II or KnowledgeMan. FGLs probably began as macros to expedite data base management procedures, but many of them have evolved into fairly complete programming languages. Just as the dBASE II language was the first to gain wide use and popularity in the microcomputer world, the SQL (pronounced “sequel”) language, developed by IBM as part of the SQL/DS DBMS, was the first to gain wide use and popularity in the mainframe world. The KnowledgeMan version of SQL is modeled after the IBM mainframe version.

When I first received SAVVY PC, I expected to find another DBMS with its accompanying FGL. Instead, I found a highly structured and modular language that uses spoken English terminology. Excalibur Technologies calls SAVVY “a natural language programming environment and set of tools for solving data processing problems [by] using natural language programming.” The Excalibur folks wrote the SAVVY PC data base manager in the SAVVY programming language (from here on referred to as SAVVY). SAVVY is not quite a true “fifth generation” language, which would allow a spoken English interface with a computer (“Hey, computer, baby, look in

the name and address data base and get me the phone number for Dr. DooBy’s ComPUter JunkYard and HamBurger EmPorium . . .”), but it’s as close to that goal as any programming language I’ve seen yet.

Pattern Recognition

Pattern recognition capabilities, one of the foundation structures of the artificial intelligence world, allow Excalibur Technologies to tout SAVVY PC as “the artificial intelligence data base.” The folks at Excalibur structured SAVVY around their Adaptive Pattern Recognition Processor (APRP). This 4K program core recognizes the commands you give SAVVY PC. If you’ve made a typing error or used improper syntax with a command, the APRP will either correct your error or suggest changes to the command by providing a menu of possible correct choices.

The SAVVY APRP is fast. Excalibur claims that it does not look up each word in the SAVVY dictionary (more about that later) as you key entries but uses an algorithmic trick to provide immediate response to entry errors. In fact, development of the APRP preceded the SAVVY language and SAVVY PC DBMS.

You may turn off the APRP and switch SAVVY from the free mode to the literal mode at any time. When you write a program with SAVVY, the manual recommends that you work in the literal mode so the APRP won’t search for words it doesn’t know. Until I got used to SAVVY, I found myself frustrated several times when I forgot to turn off the APRP. SAVVY often rewrote what I thought was a new word because the program considered my word a typing error.

The second contribution that the artificial intelligence world has made to the application software world is the natural language interface. The SAV-



PC FORTH™

IBM PC & XT,
HP-150,
Macintosh,
Apple II,
CompuPro,
Sage & CP/M-68K,
Wang PC,
All CP/M and
MSDOS computers.

Try the *professional language* offering the *utmost performance* in the *shortest development time*. Transport your applications between any of our enhanced 83-Standard compilers or expanded 32-bit versions. Choose from our wide selection of programming tools including native code compilers, cross-compilers, math coprocessor support, and B-Tree file managers. All fully supported with hotline, updates, and newsletters.

Laboratory Microsystems Incorporated

Post Office Box 10430, Marina del Rey, CA 90295

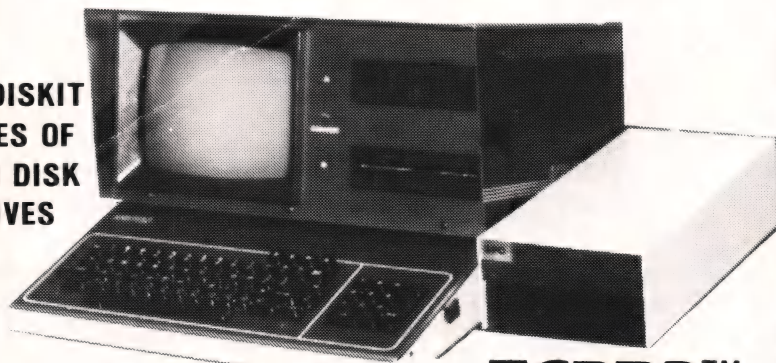
Phone credit card orders to (213) 306-7412



Circle no. 55 on reader service card.

FOR THE SERIOUS KAYPRO® USER

THE DISKIT
SERIES OF
HARD DISK
DRIVES



...now with **ZCPR3™**

Now you can add from 5 to 40 Megabytes of fast-access Winchester storage to your KAYPRO 2, 4, or 10. The DISKIT is only 4 inches high; 5.7 if you get the two drive model with the *removable* 5 or 10 Mb. cartridge, and weighs less than 10 pounds. Easily disconnect DISKIT from the computer whenever you want, and if more capacity is required, just swap your drive for a larger model.

Our DISKIT Model 10 has 10.8 Megabytes of *formatted* capacity . . . 20% more than a Kaypro 10, and runs about twice as fast. Installs in minutes. Call SPC now and ask for more information. Quantity and prepayment discounts are available.

SYSTEMS PERIPHERALS CONSULTANTS

9747 Business Park Avenue
San Diego, CA 92131
(619) 693-8611

Circle no. 104 on reader service card.

NEW FEATURES

(Free update for our early customers!)

- Edit & Load multiple memory resident files.
- Complete 8087 assembler mnemonics.
- High level 8087 support. Full range transcendentals (tan, sin, cos, arctan, logs and exponentials) Data type conversion and I/O formatting.
- High level interrupt support. Execute Forth words from within machine code primitives.
- 80186 Assembler extensions for Tandy 2000, etc.
- Video/Graphics interface for Data General Desktop Model 10

HS / FORTH

- Fully Optimized & Tested for:
IBM-PC IBM-XT IBM-JR
COMPAQ EAGLE-PC-2
TANDY 2000 CORONA
LEADING EDGE

(Identical version runs on almost all MSDOS compatibles!)

- Graphics & Text (including windowed scrolling)
- Music - foreground and background includes multi-tasking example
- Includes Forth-79 and Forth-83
- File and/or Screen interfaces
- Segment Management Support
- Full megabyte - programs or data
- Complete Assembler (interactive, easy to use & learn)
- Compare
BYTE Sieve Benchmark jan 83
HS/FORTH 47 sec BASIC 2000 sec
w/AUTO-OPT 9 sec Assembler 5 sec
other Forths (mostly 64k) 70-140 sec
FASTEST FORTH SYSTEM AVAILABLE.

**TWICE AS FAST AS OTHER
FULL MEGABYTE FORTHS!**

(TEN TIMES FASTER WHEN USING AUTO-OPT!)

HS/FORTH, complete system only: \$250.



Visa

Mastercard



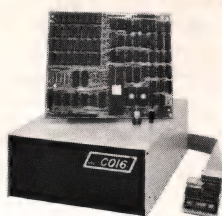
Add \$10. shipping and handling

HARVARD SOFTWARES

P.O. Box 2579
Springfield, OH 45501
513/390-2087

Circle no. 44 on reader service card.

Z80 POWER²



16/32 BIT PROCESSING POWER

In 15 minutes you can have a 16 bit O.S. running and still use your CPM80 with the touch of a key.

WE SUPPORT

CPM80 RAM DISK
MSDOS
CPM86
CPM68K
OS-9* UNIX Look-A-Like
1.25 Mb RAM**
6 Mhz No Wait States
Real Time Clock
Math Co-Processors
AND MORE

Compatible with any Z80
System Running CPM 2.2
or 3.0.

HSC's CO-1686 and CO-1668 Attached Resource Processors

Prices Starting
at \$695.



**Hallock
Systems
Company, Inc.**

Blazing the Trail

262 E. Main St.
Frankfort, NY 13340
(315) 895-7426

* Available First Quarter

** CO-1686 Expandable to 768Kb

VY RETRIEVER reportedly allows you to query a SAVVY PC data base using plain old English. I had hoped to describe how RETRIEVER works and to evaluate it for this review, but Excalibur didn't release it in time for the Dr.'s publication deadline.

The Data Base Manager

The SAVVY PC package comes with a thick, IBM-sized, detailed reference manual/slide case for SAVVY and a booklet for the SAVVY SAMPLER to introduce you to the DBMS. You also receive a master program disk and the sampler disk. Both disks contain the files SAVVY.COM, SAVVY.OVR, and DATABASE.SVY.

Excalibur wrote SAVVY PC in MMSForth (this despite John Dvorak's statement in his *InfoWorld* column, "Inside Track," that no decent program was ever written in Forth). SAVVY PC shows its Forth heritage by storing data base procedures, report formats, and the data files in DATABASE.SVY. Although the program does not use the DOS environment, DATABASE.SVY resides on a standard DOS format disk, and SAVVY PC can import and export standard DOS files. When you write a SAVVY program that doesn't use the data base manager, SAVVY PC creates a separate storage file that you name (FILENAME.SVY).

After SAVVY boots and presents its sign-on messages, it displays the prompt: "What would you like me to do now?" You may either enter the menu system by typing MAIN MENU or enter direct commands to SAVVY. The data base manager booklet and sampler disk take you through the basics of using the DBMS and generating reports. Because the DATABASE.SVY file on the sampler disk contains almost 213K, you barely have enough disk space to finish the examples before running out of storage room. SAVVY gives you adequate warning when you are about to run out of disk space. It also provides the JANITOR command to perform general disk housekeeping chores and to compact your data. However, you can avoid the disk space problem by copying DATABASE.SVY to drive B: and modifying the DEMO.BAT batch file to reflect this change.

When you tell SAVVY PC to generate a new data base, the program pro-

vides a blank form on the screen and allows you to structure the layout of your data entry screen by "painting" the screen. SAVVY PC also guesses at your data types (i.e., ANSWER, COMPUTED, DATE, GENERAL, INTEGER, MONEY, NOTE, NUMBER, PHONE, SSN, STATE, or ZIP) when you create the data entry screen. Later, the program allows you to clean up things by detailing data types, field length, and limits for data entry. You may also build relationships with other data files and specify whether data entered into your new data base will update related data bases. After you complete your design work, SAVVY PC takes from two to six minutes to generate the data base. All SAVVY's activities scroll past you as it works. Although not documented in the SAVVY manual, if you press <Ctrl><Num Lock>, you can pause SAVVY's activities to determine exactly what the program is doing to create your data base—a good tool to assist in learning the SAVVY language.

[A warning about another undocumented feature: <Ctrl><Break> exits SAVVY in a hurry. Once, while reviewing a program I was working on, I accidentally pressed <Ctrl><Break> instead of <Ctrl><Num Lock>. After some disk activity, SAVVY dumped me back to DOS. SAVVY had closed the open files, so no harm was done. However, I experimented further and found that if I hit <Ctrl><Break> while SAVVY was building a new data base, all my work would be lost.]

As time has permitted over the last six months, I have been developing a ranch management package for my wife, who is a horse trainer. I had been using KnowledgeMan (KMan), which allows much quicker and easier program development than a high-level language does, but hadn't found the time to design simplified data entry forms. KMan's traditional data base data entry format reinforced my wife's dislike of computers. She also had trouble phrasing data base queries with KMan's SQL language. After her first few frustrating attempts with the KMan ranch management package, she refused to use the program.

When I received SAVVY PC, I decided to give the ranch management

Circle no. 26 on reader service card.

project one more try—but with SAVVY PC instead of KMan. SAVVY PC simplifies the data base design process (you construct the entry screens as you structure the data base), which also speeds up the whole programming effort. I completed the SAVVY PC version of the ranch management data base in less time than it took to finish the KMan version only partially. After a great deal of coaxing, my wife tried the new program. She was pleasantly surprised. Although she doesn't dislike computers any less, at least now she uses the data base I designed for her.

SAVVY Programming

SAVVY comes with a dictionary of more than 220 words. The dictionary does not contain just the usual data base terms; it also holds all the available SAVVY commands, items, tasks, and functions. When you write a SAVVY program, you use existing SAVVY words to create new words for the dictionary; i.e., you may extend SAVVY as you would extend C or Forth. Because SAVVY words may contain up to 80 characters, you can be fairly descriptive when creating new terms.

Commands simply represent direct orders to SAVVY. However, the APRP affects the way you interact with SAVVY. For example, after you give the command COPY and press <Enter>, SAVVY responds with "COPY the _ to _." The cursor is positioned at the first underline (underlines are displayed as reverse video blocks on your CRT). After you type the name of the item you wish copied and press <Enter>, SAVVY moves the cursor to the second underline. Type the second item and <Enter>. Items may be either constants or variables; SAVVY doesn't differentiate between the two. The final result could indicate: COPY the FIRST CONSTANT to LAST VARIABLE.

Tasks and functions contain groups of commands that perform specific operations (they may also contain other tasks and functions); these groups provide SAVVY with its modular programming approach. Tasks and functions are identical except that a task stands on its own and a function requires operands or arguments to perform its chores.

For example, when SAVVY created the ranch management data base pro-

gram for my wife, I had specified that the horse's AGE field be calculated. My wife wants the data base to calculate the horse's age automatically after she enters the animal's date of birth. (By convention, all horses become one year older at the first of each year, rather than on their actual birth dates; this custom greatly simplifies the calculation.)

While watching the screen as SAV-

VY generated the ranch management data base, I saw that the program created a task called RANCH MANAGEMENT, COMPUTE THE AGE. After SAVVY finished creating the data base, I told it to detail the RANCH MANAGEMENT, COMPUTE THE AGE. Because SAVVY showed that the task contained only the line "COPY the ' to AGE," I used the SAVVY Task Editor to create the actual age calcula-

Dr. Dobbs says, "WE HAVE GOOD NEWS..."

"WINDOWS FOR C can offer you a convenient, reliable means of implementing windows in your text handling programs

"..video output is fast and clean..."

"..documented in well-written and readable English..."

"..has obviously been well thought out and very cleanly executed."

"WINDOWS FOR C is a professionally-oriented set of programming tools...that we can recommend."

—Ian Ashdown (*Dr. Dobb's Journal*, November 1984)

MORE THAN A WINDOW DISPLAY SYSTEM, WINDOWS FOR C is a video toolkit for all screen management tasks: pop-up menus and help files, status line, keyboard interpreter, word wrap, auto scroll, highlighting, color control, overlay and restore, plus a library of over 50 building block subroutines.

EASY TO LEARN. EASY TO USE. Once you've tried WINDOWS FOR C, you'll wonder how you ever managed without it.

Full support for IBM PC/XT/AT and compatibles, plus interfaces for non-IBM computers; Lattice C, CI-C86, Mark Wm. C, Aztec C, Microsoft C, DeSmet C (PC/MSDOS).



The Good News... **WINDOWS FOR C** ADVANCED SCREEN MANAGEMENT MADE EASY

A Professional Software Tool From

CREATIVE SOLUTIONS • 802 • 848-7738
21 Elm Ave., Box B • Richford, VT 05476

WINDOWS FOR C (specify compiler and version) **\$195**
Demo disk and manual (applies toward purchase) **\$30**

MasterCard & Visa Accepted
Shipping \$2.50
VT residents add 4% tax

Circle no. 27 on reader service card.

tion task as follows:

**Task RANCH MANAGEMENT ,
COMPUTE THE AGE**

```
1 COPY the DATE OF BIRTH to  
  BIRTHDATE  
2 TAKE the first 6 characters from  
  BIRTHDATE  
3 COPY the BACK to BIRTH YEAR  
4 GET DATE and time into TO-  
  DAY'S DATE  
5 TAKE the first 15 characters from  
  TODAY'S DATE  
6 COPY the BACK to THIS YEAR  
7 SUBTRACT the BIRTH YEAR  
  from THIS YEAR  
8 COPY the DIFFERENCE to AGE  
End.
```

Each time you enter a word that is not in the SAVVY dictionary (such as BIRTHDATE or BIRTH YEAR), the program informs you that it does not know the word and asks if you wish to define it. Answer "yes" and continue to enter your program. SAVVY automatically numbers the lines. Because no task or function may be more than 100 lines long, you are forced to write programs in modular blocks and to structure them in a Forth-like top down fashion.

In the above example, line 1 copies the animal's birthdate to a new item. Because you enter BIRTHDATE in the form of DD/MM/YY, line 2 can split it into the FRONT portion, DD/MM/, and the BACK portion, YY, to obtain the BIRTH YEAR. Lines 4 and 5 obtain the TODAY'S DATE from the operating system; the GET DATE command also retrieves the system time. Because this command uses a HH:MM:SS DD/MM/YY format, you must cut the TODAY'S DATE variable after the fifteenth character to obtain THIS YEAR. Line 7 performs the calculation, and line 8 copies it to the AGE.

After you finish writing a task or function, press <Alt><Q> and SAVVY will compile it. The program informs you of the nature and location of any syntax errors that the task or function may contain.

Programming in SAVVY is similar to programming with Forth. Basically, you extend SAVVY and create new tasks and functions with the existing SAVVY words, then SAVVY compiles the new tasks and functions. When your

program runs, SAVVY interprets the compiled words one by one. But unlike Forth, SAVVY's terminology is easy to remember and understand because it's plain old English. You also don't have to worry about manipulating Forth's stack or reverse Polish notation.

On the Negative Side

The only mathematical functions that SAVVY contains are add, subtract, multiply, and divide. If your programs require higher math functions, you will have to write them yourself.

SAVVY contains no way of adding comments or remarks to programs. Because of SAVVY's natural language terminology, I'm sure Excalibur Technologies didn't consider remarks necessary. But remarks often make it easier to understand long tasks or functions. However, SAVVY lends itself well to pseudocoding, which greatly simplifies program design.

Like its parent Forth, SAVVY makes no provisions for arrays and, as stated earlier, doesn't use a stack as a partial substitute for arrays. However, the program contains such extensive string-handling functions for character and bit manipulation, you probably won't need arrays. Unlike BASIC's RIGHT\$, MID\$, LEFT\$, and so on, SAVVY's string-handling commands are easy for even novice programmers to understand and use.

As massive and as well written as SAVVY's documentation is, there's just not enough of it. You must watch SAVVY build data bases to learn the details of writing programs with the language. The data base manager booklet contains only 4½ pages about extending a SAVVY-generated data base. Also, the manual contains only the particulars of programming with SAVVY and supplies no information about integrating programs into SAVVY PC-created data base programs.

A member of Excalibur Technologies informed me that a 600+ page tutorial for an earlier version of SAVVY exists. The company is updating the tutorial to meet the requirements of the present version of SAVVY and will release the book as soon as possible.

Where to Next?

Because SAVVY also contains a fairly complete set of communications com-

mands, a friend and I plan to write a remote bulletin board system (RBBS) with SAVVY. Most IBM PC RBBSs use variations of the Capitol Computer Club (CPC) RBBS, which was written in BASICA. I have seen several variations (A - D) of the CPC RBBS, version 12.3, on different systems and have also seen announcements that version 13 is imminent. An RBBS SYSOP (SYstem OPERator) recently confided in me that although CPC RBBS has been evolving for over two years, it still contains bugs. My friend and I believe that we can develop a SAVVY RBBS in just two or three months; almost any FGL, and particularly SAVVY, reduces programming time extensively.

I have pretty much stopped programming with BASIC, Turbo Pascal, and KnowledgeMan and switched to SAVVY. When the tutorial becomes available, I believe SAVVY will become one of the easiest ways to teach novice computer users about the black art of computer programming. When they reviewed SAVVY PC for the massive data base comparison series that appeared in *PC Magazine*, John F. and Barbara E. McMullen stated that programming in SAVVY PC is fun—and they were right.

DDJ



Over 40 volumes of public domain software including:

- compilers
- editors
- text formatters
- communications packages
- many UNIX-like tools

Write or call for more details

The C Users' Group

415 E. Euclid • Box 97
McPherson, KS 67460
(316) 241-1065

Circle no. 17 on reader service card.

Thanks to YOU . . . We're Growing . . .
with YOU and your Computer . . .



LEO ELECTRONICS, INC.
P.O. Box 11307
Torrance, CA 90510-1307
Tel: 213/212-6133 800/421-9565
TLX: 291 985 LEO UR

**We Offer . . . PRICE . . . QUALITY . . .
PERSONAL SERVICE**

64K UPGRADE

9 Bank (IBM PC)	\$26.10	(150ns)
	\$24.75	(200ns)
4164 (150ns)	\$2.90 ea.	
(200ns)	\$2.75 ea.	
8 Bank (other PC)	\$23.20	(150ns)
	\$22.00	(200ns)
4164 (150ns)	\$2.90 ea.	
(200ns)	\$2.75 ea.	

256K "MOTHER-SAVER" UPGRADE

256K — (150ns)	\$22.00 ea.	
6116P-3 — \$4.00	2732 — \$3.80	
2716 — \$2.95	2764 — \$5.50	
TMS-2716 — \$4.95	27128 — \$16.50	

We accept checks, Visa, Mastercard or Purchase Orders from qualified firms and institutions. U.S. funds only. Call for C.O.D. California residents add 6½% tax. Shipping is UPS. Add \$2.00 for ground and \$5.00 for air. All major manufacturers. All parts 100% guaranteed. Pricing subject to change without notice.

Electronic Circuit Analysis

- New release
- Transient, AC, DC analysis
- Full nonlinear
- Over 200 nodes
- Full editing
- Macro circuits
- Worst case, Monte-Carlo
- Temperature effects
- Frequency dependent parts
- Time dependent parts

For MS-DOS. 192k minimum.
\$395.00

Tatum Labs
33 Main Street
Newtown, CT 06470
(203) 426-2184

Circle no. 59 on reader service card.

Circle no. 82 on reader service card.

You Read Dr.Dobb's Journal And You Don't Subscribe?!

Save over \$23.00 off newsstand prices for 2 yrs.
Save over \$10.00 for 1 yr.

Can you afford to miss an issue with information vital to your interests? As a subscriber you can look forward to articles on Small-C, FORTH, CP/M, S-100, Compiler optimization, Concurrent Programming and more, delivered right to your door. And you'll never miss the issue that covers your project.

Yes! Sign me up for
_____ 2 yrs. \$47 _____ 1 yr. \$25

- _____ I enclose a check/money order
_____ Charge my Visa, MasterCard,
American Express
_____ Please bill me later

Name _____

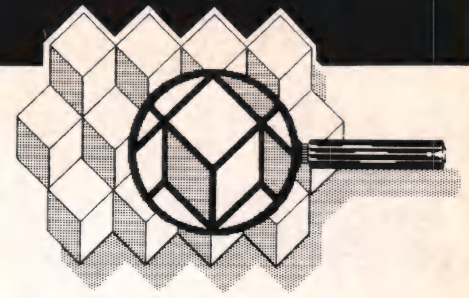
Address _____

_____ Zip _____

Credit Card _____ Exp. date _____

Account No. _____

Signature _____



by R. P. Sutherland

Turing's Man

A reading of J. David Bolter's book *Turing's Man: Western Culture in the Computer Age* has put our issue on artificial intelligence in a larger context for me. *Turing's Man* is a study of the impact that electronic logic machines are having upon our culture.

Bolter traces the history of what he calls "defining technologies" from ancient Greece until today. He does this by showing the correspondence between technologies and philosophical/theological metaphors. He points out, for example, that the Greeks, who were craftsmen, had gods who were potters and spinners. The poet who described the Fates as spinning the threads of human lives was influenced by the fact that in ancient Greece spinning was a defining technology. Plato compared the physical universe to a spindle. Likewise, during the Middle Ages the weight-driven clock provided a new metaphor for the motions of the planets. Descartes' mechanistic descriptions of the world were compelling to his contemporaries because they were familiar with clocks and gears. With the rise of the steam engine in the nineteenth century, the earth was compared to a large heat engine.

In the twentieth century's headlong rush to make a machine think like a human, we have defined the human in terms of a machine. Arguing for a synthesis of man and computer, Bolter proposes to replace the term "artificial intelligence" with "synthetic intelligence." Bolter concludes that Turing's man is the most complete synthesis of the human and the technological in Western history, and he predicts that the computer as a defining technology will give way to a technology that we cannot yet imagine. We may not realize Turing's 1950 prediction by the end of this century, but the fact that we are

thinking along these lines implies that the computer as a metaphor for the way we understand ourselves is firmly established.

Bolter has graduate degrees in the classics as well as in computer science. *Turing's Man* (ISBN 0-8078-1564-0) is published by The University of North Carolina Press, Post Office Box 2288, Chapel Hill, NC 27514. **Reader Service No. 101.**

AI

Solution Systems, the publisher of Prolog-86, is sponsoring a **contest for programmers** interested in artificial intelligence. The purpose of the contest, "Artificial Intelligence Concepts," is to generate material to help Prolog programmers and to promote Prolog. Programmers who wish to enter should develop expert systems and/or natural language processing systems designed to illustrate methods and concepts of Prolog programming. Applications with a strong commercial potential are discouraged. The application judged to be the most significant will receive a cash reward of \$1000.00. All participants will receive diskettes containing all contributions made by other contestants. Submit applications on 5¼-inch MSDOS diskettes to Solution Systems, 335 Washington Street, Norwell, MA 02061, accompanied by a one-page description. All submissions must be in by April 30, 1985. The winners will be announced in June 1985. For details call Solution Systems at (617) 659-1571. **Reader Service No. 103.**

Sun Microsystems has expanded the use of its workstations by adding AI packages to its Third Party Referral Program. Quintus Computer Systems, Inc., Palo Alto, CA, has released Quintus Prolog, which includes an interface for calling C programs. Lucid, Inc.,

Palo Alto, CA, is offering Lucid Common Lisp for developing portable knowledge-based applications on Sun Workstations. The Knowledge Engineering System (KES) from Software Architecture and Engineering, Inc., Arlington, VA, is an integrated set of utilities designed to allow newcomers to the field of AI to design expert systems. Smart System Technology, McLean, VA, offers DUCK, a high-level Lisp-based applications development environment for designing systems based on logic reasoning. JoAnn Kahn is the manager of Sun's Third Party Programs and can be reached at Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043.

Reader Service No. 105.

A Canadian company has developed a micro-version of **MPROLOG**, which, it says, opens the door to low-cost AI applications. **MPROLOG** (Modular Programming in Logic) allows a programmer to create a major program for a mainframe computer on a microcomputer by working with modules. So, by working on an IBM PC, one can develop an AI application one module at a time and execute the full application on a VAX or IBM mainframe or a 68000-based machine. **MPROLOG** for microcomputers is \$950.00 for a paid-up license or \$260.00 a month to lease. For further information: Logicware Inc., 1000 Finch Avenue West, Suite 600, Toronto, Ontario M3J 2V5 (416) 665-0022. **Reader Service No. 107.**

Software

"The arcade game is all but dead," according to Synapse Software President Jon Loveless who has just released two **electronic novels**: *Essex* and *Mindwheel*. I haven't "read" one yet, but they are supposed to put the user in the center of the action so that the user's

decisions determine the outcome of the plot. The novels are to be marketed through bookstores as well as software stores for less than \$50.00 each and are available for IBM PC, Atari, Apple IIe, Apple IIc, Macintosh, and Commodore. Synapse Software Corporation is located at 5221 Central Avenue, Richmond, CA 94804 (415) 527-7751.

Reader Service No. 109.

Speaking of novels, **RACTER** is a software program that has authored a published book: *The Policeman's Beard is Half Constructed*. The book features stories, essays, poems, and interviews created by **RACTER**. A sample poem: "More than iron, more than lead, more than gold I need electricity./I need it more than I need lamb or pork or lettuce or cucumber./I need it for my dreams." The creators, Tom Etter and Bill Chamberlain, claim that it produces original conversation that is also grammatical, without the use of stock phrases. **RACTER** is available for MSDOS or CP/M for \$69.95 from John D. Owens Associates, Inc., 12 Schubert Street, Staten Island, NY 10305 (212) 448-6298. **Reader Service No. 111.** *The Policeman's Beard is Half Constructed* (ISBN 0-446-38051-2) is published by Warner Software/Warner Books, 666 Fifth Avenue, New York, NY 10103. **Reader Service No. 113.**

A computerized thesaurus called **Synonym Finder** is available for MSDOS running WordStar or MultiMate and CP/M 80 running WordStar. **Synonym Finder** sells for \$124.95 from Writing Consultants at 11 Creek Bend Drive, Fairport, NY 14450 (716) 377-0130. **Reader Service No. 115.**

Word Publishing has announced the release of **NARNIA**, a video game based on the *Chronicles of Narnia* by C. S. Lewis. **NARNIA** is available for Apple II and Commodore 64 computers. Availability: Waldenbooks and Computerland stores, or send \$39.95 to Word Inc., Waco, TX 76703. **Reader Service No. 117.**

Thoroughbred Handicapper II is a computerized system that analyzes thoroughbred horse races and helps to predict the winners. The handicapping system is based on a statistical analysis of horse races. **Thoroughbred Handicapper II** is available for the IBM PC family for \$49.95. Contact The Software Bottling Company of New York,

29-14 23rd Avenue, Long Island City, NY 11105 (718) 728-2200. **Reader Service No. 119.**

A **brain wave to electronic speech system** for the ZX-81/TS-1000 and Apple computers permits brain wave control over robotic devices. We cannot verify this claim, but for more information write to Rosetronix, P.O. Box 7464, Chula Vista, CA 92012.

Reader Service No. 121.

Hardware

The **Integral Personal Computer** from Hewlett-Packard is a 68000 Unix-based machine. This portable computer has a built-in ThinkJet printer, a 3½-inch (710K) disc drive, a 9-inch electroluminescent display, and a full-size keyboard. Price: under \$5,000.00. Hewlett-Packard, P.O. Box 10301, Palo Alto, CA 94303. **Reader Service No. 123.**

WHY WOULD ANY SANE PERSON SPEND \$199 FOR A BetterBASIC SYSTEM WHEN DOS's IS FREE?

HERE ARE 10 REASONS:
TEST YOUR SANITY

1. Full support for 640K memory
2. Structured language with BASIC syntax
3. Separately compiled program modules
4. Speed: FAST
5. Extensibility (Make your own BASIC.)
6. User-defined procedures and functions
7. Built-in windows support
8. Interactive programming language based on an incremental compiler
9. 8087 math support
10. Runs on IBM PC, IBM PC/XT and compatibles

Summit Software
Technology, Inc.™
P.O. Box 99
Babson Park
Wellesley, MA 02157
1-800-225-5800

BetterBASIC is a trademark of Summit Software Technology, Inc. IBM PC, IBM PC/XT and PC/DOS are trademarks of International Business Machines Corp. MS-DOS is a trademark of Microsoft Corp.

**NOW AVAILABLE FOR
THE TANDY 2000 & 1200**

**Better
BASIC™**

**Sane Programmers
Order BetterBASIC Now**

Price: \$199
8087 Math Module: \$99
Runtime System: \$250
Sample Disk: \$10

MasterCard, VISA, P.O. Checks,
Money Orders, and
C.O.D. accepted.



Are You Working With a Full Deck?

Increase your chances of a winning hand.

Confront today's issues of multi-user, multi-tasking computer systems and collect your chips tomorrow!

Use this pair to complete the deal.



These key computer expositions promise a full house in two of the major computer technology centers — San Francisco and Boston.

UNIX Systems Expo/85 — Spring
April 24-26 San Francisco

UNIX Systems Expo/85 — Fall
October 3-5 Boston

Exhibit:

if your market consists of large corporate users, VARs, software developers, systems integrators and OEMs.

Attend:

to explore and understand the emerging multi-user, multi-tasking computer systems and products.

Exclusive productions of
Computer Faire, Inc./a Prentice-Hall
Company

181 Wells Avenue
Newton, MA 02159
617/965-8350

611 Veterans Boulevard
Redwood City, CA 94063
415/364-4294

*UNIX is a trademark of AT&T Bell
Laboratories

Yes, I'm interested in hearing more about
UNIX* Systems Expo/85. Please send
me ☐ exhibit/ ☐ attendee
information on:

☐ San Francisco
☐ Boston
☐ Both

Name

Title

Company

Address

City/State/Zip

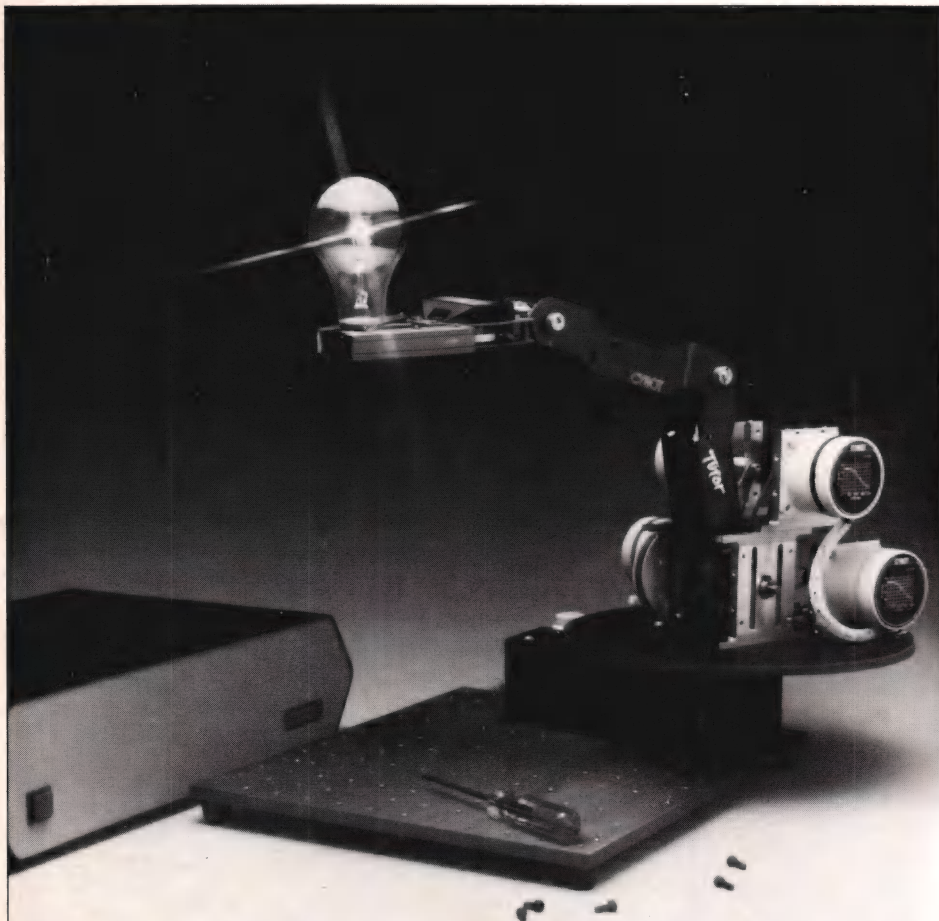
Phone

(D)

Computer Faire, Inc. 181 Wells Avenue Newton, MA 02159



The Spectravideo **Bondwell 12** computer runs CP/M 2.2 on a Z80a CPU. The price of \$995.00 includes a voice synthesizer, two floppy disk drives, and bundled software. Christopher Chang, President, Spectravideo, 3300 Seldon Court #10, Fremont, CA 94539 (415) 490-4300. **Reader Service No. 125.**



A modular robot called **Tutor** is available as a kit from Cybot, Inc. for \$3,395.00. The three circuit boards used to control the Tutor robot are designed to the S-100 standard. Cybot, Inc., 733 7th Avenue, Kirkland, WA 98033. **Reader Service No. 129.** DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service No. 197.

DeSmet C

**8086/8088
Development
Package**

\$109

FULL DEVELOPMENT PACKAGE

- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full-Screen Editor
- Execution Profiler
- Complete **STDIO** Library (>120 Func)

Automatic DOS 1.X/2 X SUPPORT

**BOTH 8087 AND S/W FLOATING POINT
OVERLAYS**

OUTSTANDING PERFORMANCE

- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER **\$50**

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT **\$35**

- Uses DOS .OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

Check: ☐ Dev. Pkg (109)
☐ Debugger (50)
☐ DOS Link Supt (35)

SHIP TO: _____

CW ARE
CORPORATION

**P.O. BOX C
Sunnyvale, CA 94087
(408) 720-9696**

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add \$5. elsewhere add \$15. Checks must be on US Bank and in US Dollars. Call 9 a.m. - 1 p.m. to CHARGE by VISA/MC/AMEX.

Circle no. 18 on reader service card.

125

DDJ Classifieds

Software

PCBTAM

Communications Access Method

Allows an IBM PC or compatible to perform BISYNC communication. PCBTAM is a general purpose interrupt driven access method usable by any Microsoft language. Requires IBM BSCA card and PC-DOS.

- Source or object license
- Asynchronous version available (PCATAM)
- Modifiable for other USARTS.

For details write:

SYMBIOTIC

Symbiotic Protocol Converters, Inc.
1011 Clifton Avenue, Clifton, New Jersey 07013
(201) 777-6454

Olive Branch Software

COPY PROTECTION

SLK/F places an assembled or compiled program on a diskette with 4 different copy-resistant features in such a way that it runs normally, but cannot be copied by backup programs such as COPYPC. The rest of the diskette is available as normal, and DOS may be added. Price \$150.

Olive Branch Software
1715 Olive Street
Santa Barbara, CA 93101
(805)569-1682

Offering low cost advertising reaching thousands of computer programmers and professionals each month. Departments to choose from include:
Software
Hardware
Accessories/Supplies
Career Opportunities
Services
User Groups. Special categories are also available.

Announcing . . .

Dr. Dobb's Journal Bound Volume 7

Includes every 1982 issue of Dr. Dobb's Journal

Yes! Please send me Volume #7

I enclose ☐ check/money order.

Please charge my: ☐ VISA ☐ M/C ☐ American Express

Card # _____ Expiration Date _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

Volume 7 _____ x \$30.75 = \$ _____

Payment must accompany your order. Mail to Dr. Dobb's Journal.
2464 Embarcadero Way, Palo Alto, CA 94303.
Allow 6-9 weeks for delivery.

Postage & Handling must be included with your order.
Please add \$1.25 per book in U.S., \$2.00 each outside U.S.

☐ Please send me more information on other Bound Volumes.

Dr. Dobb's Journal is pleased to announce the DDJ Classifieds

RATES: DISPLAY ADVERTISERS: Price per column inch \$100. Ad must run in 3 consecutive issues.

LINE ADVERTISERS: Price per line \$12 (35 characters per line including spaces). Minimum of 5 lines. 3 consecutive issues. Add \$3 per line for boldface type. Add \$25 if a background screen is desired.

DISCOUNTS: Frequency discounts for 6 consecutive ads (less 7%) and for 12 consecutive ads (less 15%).

MECHANICAL REQUIREMENTS: Camera ready art or typewritten copy only (phone orders excepted). Display: Specify desired size, include \$20 if reduction is required. Line: Specify characters in boldface, caps. Allow 6 weeks for publication.

PAYMENTS: Full payment in advance. Check, money order, Visa, M/C, AmEx.

Run this ad in _____ issues

Size: _____ col x _____ inches or 1 col x _____ lines

Payment by: _____ check _____ m/o _____ Visa _____ M/C _____ AmEx

Card # _____ Exp Date _____

Signature _____

Print Name _____

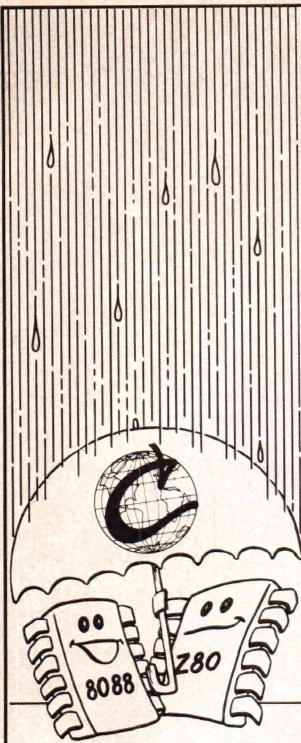
Company _____

Address _____

City _____ St _____ Zip _____

Phone _____ Current Subscriber _____

Mail to or phone Alex Williams, DDJ Classifieds, 2464 Embarcadero Way, Palo Alto, CA 94303 (415) 424-0600



Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

An Optimizing, hassle-free C Compiler for the 8086-8088.

Ecosoft's Eco-C, the performance leader among full C compilers for the Z80, is now available for the 8086-8088 running under MSDOS (2.0 or later). Eco-C has features not found in any other 8086-8088 C compiler.

★ Over 100 library functions. Since they follow UNIX standards, your programs are highly portable in "both" directions ("up" to a UNIX machine or "down" to our Z80 compiler). This means new markets for your software at minimum development cost.

★ A single library. No more "dual" libraries and trying to remember what has to be linked with what. Your programs automatically take advantage of an 8087 if one is present at runtime.

★ A single floating point answer. No more "fuzzy floating point": your programs produce the same answers whether the floating point is done in hardware (8087) or software.

★ Error messages in English. No more cryptic numbers to look up. We tell you where the error occurred, what was found there, and what should have been there.

★ Strict syntax parsing. LINT is going to uncover fewer surprises because our parser looks hard at the details.

★ Efficient code. The optimizer pass of the compiler generates assembler code in Intel mnemonics that rivals that produced by compilers costing twice as much.

The price of the Eco-C compiler is \$250.00 (all versions), including the user's manual, and is designed for use with Microsoft's MASM (or compatible) assembler and linker. When ordering, please specify disk format and whether you want the Z80-CP/M or 8088-MSDOS version of Eco-C.

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476



Circle no. 35 on reader service card.

DSD 80

FULL SCREEN SYMBOLIC DEBUGGER

**"THE SINGLE BEST DEBUGGER
FOR CP/M-80. A TRULY
AMAZING PRODUCT."**

LEOR ZOLMAN
AUTHOR OF BDS C

- ☐ Complete Upward Compatibility with DDT
- ☐ Simultaneous instruction, register, stack & memory displays
- ☐ Software in-circuit-emulator provides write protected memory, execute only code and stack protection.
- ☐ Fifteen single keystroke commands
- ☐ Thirty day money back guarantee
- ☐ On-line help & 50 page user manual

ONLY \$195.

IBM PC VERSION AVAILABLE SOON!

SOFTADVANCES

P.O. BOX 49473 AUSTIN, TEXAS 78765 (512) 478-4763



Circle no. 63 on reader service card.

C

Software Development PCDOS/MSDOS

Complete C Compiler

- Full C per K&R
- Inline 8087 or Assembler Floating Point, Auto Select of 8087
- Full 1Mb Addressing for Code or Data
- Transcendental Functions
- ROMable Code
- Register Variables
- Supports Inline Assembler Code

MSDOS 1.1/2.0 Library Support

- All functions from K&R
- All DOS 2.0 Functions
- Auto Select of 1.1 or 2.0
- Program Chaining Using Exec
- Environment Available to Main

c-window™

Symbolic Debugger

- Source Code Display
- Variable Display & Alteration Using C Expressions
- Automatic Commands
- Multiple Breakpoints by Function & Line Number

8088/8086 Assembler

- FAST — Up to 4 times Faster than IBM Assembler
- Standard Intel Mnemonics
- Compatible with MSDOS Linker
- Supports Full Memory Model

8088 Software Development Package

\$199⁰⁰

Includes: C Compiler/Library, c-window, and Assembler, plus Source Code for c-systems Print Utility

c-systems

P.O. Box 3253
Fullerton, CA 92634
714-637-5362

Circle no. 16 on reader service card.

C Programmers: Program three times faster with *Instant-C*TM

*Instant-C*TM is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use as Basic. Yet *Instant-C*TM is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into *Instant-C*TM for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

Instant Execution. *Instant-C*TM uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Symbolic Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Floating Point. Uses 8087* co-processor if present.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Get more done, faster, with better results. *Instant-C*TM is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*.

Find out how *Instant-C*TM is changing the way that programming is done.

*Instant-C*TM is \$500. Call or write for more information.

Rational
Systems, Inc.

(617) 653-6194

P.O. Box 480

Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intell Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

Circle no. 79 on reader service card.

ADVERTISER INDEX

Reader Service No.	Advertiser	Page No.	Reader Service No.	Advertiser	Page No.
5	A/N Software Inc.	49	62	MicroMotion	69
1	Alpha Computer Service	15	41	MicroSmith Computer Tech.	115
7	Ampro Computers Inc.	93	64	Microprocessors Unlimited	115
11	Anthropomorphic Systems	110	66	Mitek	73
8	Apropos Technology	49	*	Mix Software	C3
4	Ashton Tate	24-25	51	Morgan Computing Co.	101
10	Avocet Systems, Inc.	35	56	Motel Computers Ltd.	115
12	BD. Software, Inc.	99	91	Mullen Computer Products	15
13	B.G. Micro	59	126	Multi-Tech Electronics	39
14	Borland International	C-4	76	New Generation Systems	85
16	C Systems	127	60	Nicolet Paratronics	111
17	C User's Group	120	61	Northwest Computer Algorithms	103
18	C Ware	115	68	Parsec Research	85
20	Catspaw	115	70	Phoenix Computer Products	11
6	Central Point Software	73	71	Poor Persons Software	77
21	Chalcedony	71	73	Procode International	91
23	CompuServe	C-2	67	The Programmer's Shop	55
9	Compucraft	115	87	Programming Logic Systems, Inc.	32
15	Computer Faire, Inc.	124	83	Raima Corp.	29
22	Computer Helper Industries, Inc.	103	79	Rational Systems, Inc.	128
27	Creative Solutions	119	80	Revasco	93
28	D&W Digital	33	85	SemiDisk Systems	93
30	Data Base Decisions	97	86	Shaw American Technologies	105
31	Datalight	85	106	Signum Systems	110
33	Digital Research Computers	7	63	Soft Advances	127
29	Diversified Educational Enterprises	100	88	Softaid, Inc.	115
35	Ecosoft, Inc.	127	89	Softfocus	115
*	Edward Ream	105	90	Software Horizons, Inc.	81
114	Emerald Systems	2	69	Software Information Systems	94
36	Essential Software	12	92	Softway, Inc.	81
37	Faircom	91	75	Solution Systems	57
40	Fox Software Inc.	31	93	Solution Systems	57
38	Gimpel Software	41	94	Solution Systems	57
43	Greenleaf Software, Inc.	13	95	Solution Systems	57
26	Hallock Systems Consultants	118	97	Speedware	69
44	Harvard Softworks	117	65	Spruce Technologies Corp.	91
46	IQ Software	19	110	Starlight Forth Systems	69
48	Illyes Systems	29	98	Summit Software	123
*	Integral Quality	73	104	Systems Peripheral Consultants	117
50	Interface Technologies	42-43	82	Tatum Labs	121
24	J & M Systems, Ltd	51	77	UniPress Software	17
	J. D. Owens Associates	77	108	Unlimited Processing	95
55	Laboratory Microsystems Inc.	117	102	Vance Info Systems	71
57	Lark Software	110	78	Vertex Systems, Inc.	49
58	Lattice, Inc.	29	112	Wendin, Inc.	9
25	Lavine Instrument Company	115	118	Western Wares	115
59	Leo Electronics, Inc.	121	116	Wizard Systems	15
53	Level 5 Research	81	*	DDJ Back Issues	98
74	Lifeboat Associates	111	*	DDJ Bound Volume/Reston	106-107
54	LogicWare, Inc.	1	*	DDJ C Compiler	111
84	Megamax, Inc.	103	*	DDJ Classifieds	126
32	Micro Cornucopia, Inc.	3	*	DDJ Subscription	121
39	MicroDynamics Corp.	103	*	DDJ Bound Volume	61

Introducing the MIX Editor

A Powerful Addition To Any Programmer's Tool Box

Full Screen Editing
WordStar Key Layout
Custom Key Layouts
Terminal Configuration
Help Files
Backup Files

*Introductory Offer
Only*

2995

30 Day Money Back Guarantee

Programmable
Macro Commands
Custom Setup Files
Mnemonic Command Mode
Multiple File Editing
Unlimited File Size

For PCDOS/MSDOS (2.0 and above/128K) or CPM80 (Z80 required/64K)

Great For All Languages

A general purpose text processor, the MIX Editor is packed with features that make it useful with any language. It has auto indent for structured languages like Pascal or C. It has automatic line numbering for BASIC. It even has fill and justify for English.

Terminal Configuration

A utility for defining terminal features (smart features included) allows the editor to work with any terminal. Over 30 of the most popular terminals are built-in.

Custom Keyboard Layout

Commands are mapped to keys just like WordStar. If you don't like the WordStar layout, simply change it. Any key can be mapped to any command. You can also define a key to generate a string of characters, great for entering keywords.

Macro Commands

The MIX Editor allows a sequence of commands to be executed with a single keystroke. You can define a complete editing operation and perform it at the touch of a key.

MIX 2116 E. Arapaho
Suite 363
Richardson, Tx 75081
software (214) 783-6001

MSDOS is a trademark of Microsoft
PCDOS is a trademark of IBM
CPM80 is a trademark of Digital Research
WordStar is a trademark of MicroPro

Custom Setup Files

Custom keyboard layouts and macro commands can be saved in setup files. You can create a different setup file for each language you use. The editor automatically configures itself using a setup file.

Command Mode

Command mode allows any editor command to be executed by name. It is much easier to remember a command name versus a complicated key sequence. Command mode makes it easy to master the full capability of the editor. Frequently used commands can be mapped to keys. Infrequent commands can be executed by name.

Editor Commands

The editor contains more than 80 commands. With so many commands, you might think it would be difficult to use. Not so, it is actually extremely simple to use. With command mode, the power is there if you need it, but it doesn't get in your way if you don't. Following is a list of some of the commands.

Cursor Commands

Left/Right/Up/Down
Tab Right/Tab Left
Forward Word/Backward Word
Beginning of Line/End of Line
Scroll Up/Scroll Down
Horizontal Scroll
Top of Buffer/Bottom of Buffer

Block Commands

Copy/Insert/Delete
Lower Case/Upper Case
Fill/Justify
Print/Extract

File Commands

Directory
Show File
Insert File
Delete File
Append File

Other Commands

Find String/Replace String
Replace Global
Delete Line/Insert Line
Undelete Line
Delete Character/Delete Word
Insert Mode
Split Line/Merge Line
Duplicate Line
Center Line
Set Tab/Clear Tab

To Order: Call Toll Free 1-800-622-4070, (Illinois only 1-800-942-7317)

Mix Editor ____ \$29.95 + shipping (\$5 USA/\$10 Foreign) Texas residents add 6% sales tax

Visa ____ MasterCard ____ Card # ____ Exp. Date ____

COD ____ Check ____ Money Order ____ Disk Format ____

Computer ____ Operating System: MSDOS ____ PCDOS ____ CPM80 ____

Name ____

Street ____

City/State/Zip ____

Country ____

Phone ____

MIX 2116 E. Arapaho
Suite 363
Richardson, Tx 75081
software
Dealer Inquiries Welcome
Call (214) 783-6001

They said it couldn't be done. Borland Did It. Turbo Pascal 3.0

The industry standard

With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

COMPILATION SPEED	206 sec
EXECUTION SPEED	20 sec.
CODE SIZE	35K
BUILT-IN INTERACTIVE EDITOR	NO
ONE STEP COMPILE (NO LINKING NECESSARY)	NO
COMPILER SIZE	300K+
TURTLE GRAPHICS	NO
BCD OPTION	NO
PRICE	\$295.00

MS PASCAL TURBO 2.0 TURBO 3.0

206 sec	16 sec.	8 sec.
20 sec.	13 sec.	9 sec.
35K	12K	12K
NO	YES	YES
NO	YES	YES
300K+	35K	35K
NO	NO	YES
NO	NO	YES
NO	\$49.95	\$69.95
\$295.00		

The best just got better: Introducing Turbo Pascal 3.0

We just added a whole range of exciting new features to Turbo Pascal:

- First, the world's fastest Pascal compiler just got faster. Turbo Pascal 3.0 compiles twice as fast as Turbo Pascal 2.0! No kidding.
- Then, we totally rewrote the file I/O system, and we also now support I/O redirection.
- For the IBM PC versions, we've even added "turtle graphics" and full tree directory support.
- For all 16 Bit versions, we now offer two additional options: 8087 math coprocessor support for intensive calculations and Binary Coded Decimals (BCD) for business applications.
- And much much more.

The Critics' Choice

Jeff Duntemann, PC Magazine: "Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."

Dave Garland, Popular Computing: "Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 29K bytes of random-access memory."

Jerry Pournelle, BYTE: "What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."

Portability

Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX version of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

An Offer you Can't Refuse

Until June 1st, 1985, you can get Turbo Pascal 3.0 for only \$69.95. Turbo Pascal 3.0, equipped with either the BCD or 8087 options, is available for an additional \$39.95 or Turbo Pascal 3.0 with both options for only \$124.95. As a matter of fact, if you own a 16 Bit computer and are serious about programming, you might as well get both options right away and save almost \$25.

Update policy

As always, our first commitment is to our customers. You built Borland and we will always honor your support.

So, to make your upgrade to the exciting new version of Turbo Pascal 3.0 easy, we will accept your original Turbo Pascal disk (in a bend-proof container) for a trade-in credit of \$39.95 and your Turbo87 original disk for \$59.95. This trade-in credit may only be applied toward the purchase of Turbo Pascal 3.0 and its additional BCD and 8087 options (trade-in offer is only valid directly through Borland and until June 1st, 1985).

11

(*) Benchmark run on an IBM PC using MS Pascal version 3.2 and the DOS linker version 2.6. The 179 line program used is the "Gauss-Seidel" program out of Alan R. Miller's book: *Pascal programs for scientists and engineers* (Sybex, page 128) with a 3 dimensional non-singular matrix and a relaxation coefficient of 1.0.

TURBO PASCAL

NOT COPY PROTECTED

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

YES! I want the Best! Quantity

Please send:

Pascal 3.0	\$ 69.95	_____
Pascal w/8087	\$109.90	_____
Pascal w/BCD	\$109.90	_____
Pascal w/8087 & BCD	\$124.95 (SAVE \$24.90)	_____

*These prices include shipping to all U.S. cities. All foreign orders add \$10 per product ordered.

Trade-in Credit Claimed: _____

Amount: (CA 6% tax) _____

Payment: VISA MC BankDraft Check _____

Credit Card Expir. Date: _____

Name on Card: _____

Card #: _____

For update: original Turbo disk must accompany order

Carefully Describe your Computer System!

Mine is: ☐ 8 bit ☐ 16 bit MS-DOS
I Use: ☐ PC-DOS ☐ CP/M 80 CP/M 86
My computer's name/model is: _____

The disk size I use is:
☐ 5 1/4" ☐ 8" ☐ 3 1/2"

Name: _____

Shipping Address: _____

City: _____ Zip: _____

State: _____

Telephone: _____

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add \$10 and make payment by bank draft, payable in US dollars drawn on a US bank.



Software's Newest Direction
4113 Scotts Valley Drive
Scotts Valley, California 95066
TELEX: 172373

Turbo Pascal is a registered trademark of Borland International, Inc.